▼ AMSTRAD ▼ SPECTRUM ▼ COMMODORE ▼ IBM ▼ MSX ▼

Informática y programación

11

PASO A PASO



PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼





PROGRAMAS EDUCATIVOS
PROGRAMAS DE UTILIDAD
PROGRAMAS DE GESTION
PROGRAMAS DE JUEGOS

▼ BASIC ▼ MAQUINA ▼ PASCAL ▼ LOGO ▼ OTROS LENGUAJES ▼
▼ TECNICAS DE ANALISIS Y DE PROGRAMACION ▼

▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼

Una publicación de

EDICIONES SIGLO CULTURAL, S.A.

Director-editor:
RICARDO ESPAÑOL CRESPO.
Gerente:
ANTONIO G. CUERPO.
Directora de producción:
MARIA LUISA SUAREZ PEREZ.
Directores de la colección:
MANUEL ALFONSECA, Doctor Ingeniero de Telecomunicación y Licenciado en Informática.
JOSE ARTECHE, Ingeniero de Telecomunicación.
Diseño y maquetación:
BRAVO-LOFISH.
Fotografía:
EQUIPO GALATA.
Dibujos:
JOSE OCHOA

TECNICAS DE PROGRAMACION: Manuel Alfonseca, Doctor Ingeniero de Telecomunicación y Licenciado en Informática. TECNICAS DE ANALISIS: José Arteche, Ingeniero en Telecomunicación. LENGUAJE MAQUINA 8086: Juan Rojas Licenciado en Ciencias Físicas e Ingeniero Industrial. PASCAL: Juan Ignacio Puyol, Ingeniero Industrial. PROGRAMAS (educativos, de utilidad, de gestión y de juegos): Francisco Morales, Técnico en Informática y colaboradores. Coordinador de AULA DE INFORMATICA APLICADA (AIA): Alejandro Marcos, Licenciado en Ciencias Químicas. BASIC: Esther Maldonado, Diplomada en Arquitectura. INFORMATICA BASICA: Virginia Muñoz, Diplomada en Informática. LENGUAJE MAQUINA Z-80: Joaquín Salvachúa, Diplomado en Telecomuniación y José Luis Tojo, Diplomado en Telecomunicación. LENGUAJE MAQUINA 6502: (desde el tomo 5): Juan José Gómez, Licenciado en Química. LOGO: Cristina Manzanero, Licenciada en Informática. APLICACIONES: Soledad Tamariz, Diplomada en Telecomunicación. OTROS LENGUAJES (sistemas operativos): Domingo Villaseñor, Diplomado en Informática, y Lenguaje C: Enrique Serrano, Ingeniero en Telecomunicación.

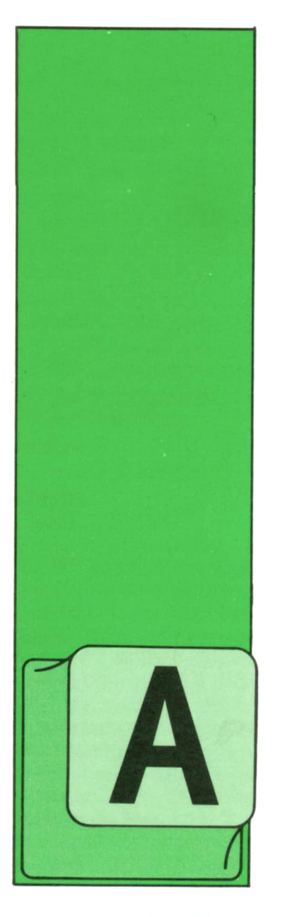
Ediciones Siglo Cultural, S.A.
Dirección, redacción y administración:
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Publicidad:
Gofar Publicidad, S.A. Benito de Castro, 12 bis. 28028 Madrid.
Distribución en España:
COEDIS, S.A. Valencia, 245. Teléf. 215 70 97. 08007 Barcelona.
Delegación en Madrid: Serrano, 165. Teléf. 411 11 48.
Distribución en Ecuador: Muñoz Hnos.
Distribución en Perú: DISELPESA.
Distribución en Chile: Alfa Ltda.
Importador exclusivo Cono Sur:
CADE, S.R.L. Pasaje Sud América, 1532. Teléf.: 21 24 64.
Buenos Aires - 1.290. Argentina.

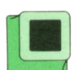
ISBN del tomo: 84-7688-103-7.
ISBN de la obra: 84-7688-068-7
Fotocomposición:
ARTECOMP, S.A. Albarracín, 50. 28037 Madrid.
Imprime:
MATEU CROMO. Pinto (Madrid).

Depósito legal: M-5-677-1987
Printed in Spain - Impreso en España.
Suscripciones y números atrasados:
Ediciones Siglo Cultural, S.A.
Pedro Teixeira, 8, 2.ª planta. Teléf. 810 52 13. 28020 Madrid.
Mayo, 1987.
P.V.P. Canarias: 335,–.

INDICE

BASIC



 IF-THEN-ELSE

ALGUNAS versiones del BASIC permiten la utilización de una sentencia de bifurcación condicional más potente: IF-THEN-ELSE, que tiene el siguiente formato:

IF <condición> THEN <lista de instrucciones 1>
ELSE <lista de instrucciones 2>

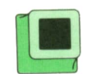
Si se verifica la *condición* indicada entre IF y THEN, se ejecutará la *lista de instrucciones 1* y, en caso contrario, la *lista de instrucciones 2*.

En el programa 1 podemos ver un ejemplo:

```
10 REM *************
20 REM *  NUMEROS  *
30 REM *************
40 CLS
50 INPUT "DIME UN NUMERO ";N
60 PRINT :PRINT
70 IF N=0 THEN PRINT "CERO":END
80 IF N>0 THEN PRINT "POSITIVO"
   ELSE PRINT "NEGATIVO"
```

Si el número tecleado es cero, se verificará la condición de la línea 70 y, como hay un END, al final, la línea 80 no se ejecutará. Sin embargo, si tecleamos cualquier otro número, la ejecución pasará a la línea 80, donde, si el número tecleado es mayor que cero, se verificará la condición y, por tanto, se imprimirá en pantalla POSITIVO. Si, por el contrario, no se cumple la condición, es decir, el número tecleado es menor que cero, entonces se imprimirá en pantalla NEGATIVO.

Las máquinas en estudio, excepto el COMMODORE y el SPECTRUM, disponen de la instrucción ELSE.

 Condiciones compuestas. Operadores lógicos

Hasta ahora las condiciones vistas en la instrucción IF-THEN eran condiciones simples; sin embargo, podemos utilizar condiciones tan complejas como deseemos haciendo uso de los *operadores lógicos*, también llamados *operadores booleanos* por estar basadas en el álgebra de Boole.

Los operadores lógicos son los siguientes:

NOT
AND
OR
XOR
EQU
IMP

Una condición compuesta es aquélla formada por dos o más condiciones simples ligadas entre sí por un operador lógico (NOT es el único que cumple esta norma general).

Por tanto, podríamos formar condiciones compuestas tan complejas como, por ejemplo, ésta:

2 < 3 AND NOT 3 + 2 = 6 OR 4/2−1>=−3

Realmente es prácticamente imposible que lleguemos a encontrar alguna vez una condición tan extraña. Sin embargo, es conveniente saber cómo analiza el ordenador una condición de este tipo para saber si es verdadera o falsa. Ya vimos que los operadores aritméticos seguían unas prioridades en el orden de las operaciones; pues bien, en la tabla de la figura 1 podemos ver la prioridad de todos los operadores (aritméticos, de relación y lógicos).

Tipo	Operador	Prioridad
Paréntesis	( )	1
Aritméticos	↑ * / + −	2 3 4
De relación	= < > < ⊥ < = > =	5
Lógicos	NOT AND OR XQR/EQU/IMP	6 7 8 9


Prioridades de los operadores.

Si analizamos la condición puesta de ejemplo siguiendo esta tabla podemos comprobar que es verdadera.

A continuación vamos a analizar el efecto que produce cada uno de los operadores lógicos:

NOT

Es el único que no se emplea para conectar dos condiciones simples, sino que siempre va asociado a una única condición simple. Su efecto es invertir el resultado de la condición a la que afecta de modo que si la condición es verdadera, NOT la transforma en falsa y viceversa.

Por ejemplo:

NOT A=B

será cierto si A es distinto de B, mientras que si A y B son iguales, la condición será falsa. De aquí se deduce que la condición compuesta NOT A=B es equivalente a la condición simple A<>B.

AND

Conecta dos o más condiciones simples. La condición compuesta resultante sólo es cierta si lo son cada una de las condiciones simples que la forman.

OR

Conecta dos o más condiciones simples. Para que la condición compuesta resultante sea cierta basta con que lo sea al menos una de las condiciones sencillas que la integran.

XOR

Conecta dos o más condiciones sencillas. La condición global será cierta cuando lo sea una sola de las condiciones elementales.

EQU

Conecta dos o más condiciones sencillas. La condición compuesta será cierta cuando las condiciones simples que la integran sean o bien todas ciertas o bien todas falsas.

IMP

Conecta dos o más condiciones sencillas. La condición resultante será cierta siempre que la primera condición elemental sea falsa.

En la figura 2 podemos ver las tablas de verdad de los cinco últimos operadores lógicos, que pueden servir para aclarar el efecto de cada uno de ellos.

Condición-Simple-1	Condición-Simple-2	Cond.-Simple-1 AND Cond.-Simple-2
V	V	V
V	F	F
F	V	F
F	F	F

Cond.-Simple-1	Cond.-Simple-2	Cond.-Sim.-1 OR Cond.-Sim.-2
V	V	V
V	F	V
F	V	V
F	F	F

Cond.-Simple-1	Cond.-Simple-2	Cond.-Sim.-1 XOR Cond.-Sim.-2
V	V	F
V	F	V
F	V	V
F	F	F
Cond.-Simple-1	**Cond.-Simple-2**	**Cond.-Sim.-1 EQU Cond.-Sim.-2**
V	V	V
V	F	F
F	V	F
F	F	V
Cond.-Simple-1	**Cond.-Simple-2**	**Cond.-Sim.-1 IMP Cond.-Sim.-2**
V	V	V
V	F	F
F	V	V
F	F	V



Tablas de verdad de los operadores lógicos. V = verdadero. F = falso.

A continuación vamos a desarrollar algunos programas como ejemplos de aplicación de las condiciones compuestas.

El programa 2 sirve para determinar si un alumno ha aprobado o no una asignatura, dependiendo de las notas que haya obtenido en tres exámenes parciales. La condición para aprobar la asignatura es haber aprobado cada uno de los tres exámenes independientemente, con una nota mayor o igual que 5.

```
10 REM **************
20 REM *  EXAMENES  *
30 REM **************
40 CLS
50 INPUT "TECLEE LAS NOTAS DE LOS TRES
   PARCIALES ";N1,N2,N3
60 CLS
70 PRINT "EXAMEN";TAB(20);"NOTA"
80 PRINT
90 PRINT "PRIMER PARCIAL";TAB(20);N1
100 PRINT "SEGUNDO PARCIAL";TAB(20);N2
110 PRINT "TERCER PARCIAL";TAB(20);N3
120 PRINT :PRINT
130 PRINT "CALIFICACION FINAL"
140 PRINT "------------------"
150 IF N1>=5 AND N2>=5 AND N3>=5 THEN
    PRINT "APROBADO":END
160 PRINT "SUSPENSO"
```

Para que la condición compuesta de la línea 150 se verifique es necesario que se verifiquen cada una de las tres condiciones simples que la forman; por tanto, tenemos que utilizar el operador AND. En caso de no cumplirse alguna de las condiciones sencillas, la ejecución del programa pasará a la línea 160, donde se imprimirá en pantalla SUSPENSO.

Por otra parte, en este programa se ha cuidado especialmente la presentación en pantalla, ya que para que un programa sea bueno no basta con que funcione correctamente, sino que es conveniente que además la presentación en pantalla sea atractiva y ordenada. En la figura 3 podemos ver el resultado de una posible ejecución.

Examen	Nota
Primer parcial	6
Segundo parcial	5
Tercer parcial	7
Calificación final	
Aprobado	
OK	



Presentación en pantalla del programa 2 (EXAMENES).

Por otra parte, podemos observar que el programa funcionaría exactamente igual si cambiáramos la condición compuesta por su contraria.

Para comprobar esto sólo tenemos que cambiar la línea 150 por la siguiente:

```
150 IF N1<5 OR N2<5 OR N3<5
THEN PRINT "SUSPENSO":END
```

y la línea 160 por:

```
160 PRINT "APROBADO"
```

También podríamos sustituir ambas líneas por una única línea de la forma siguiente:

150 IF N1<5 OR N2<5 OR N3<5 THEN PRINT "SUSPENSO" ELSE PRINT "APROBADO"

El programa 3 es un ejemplo del uso de condiciones compuestas con datos alfanuméricos. Supondremos que el programa funciona el día de San Joaquín y Santa Ana y su objetivo es felicitar a las personas que sea su santo.

```
10 REM ******************
20 REM *  FELICITACION  *
30 REM ******************
40 CLS
50 INPUT "¿COMO TE LLAMAS? ";N$
60 PRINT :PRINT
70 IF N$="JOAQUIN" OR N$="ANA" THEN
   PRINT "FELICIDADES ";N$;", YA SE
   QUE HOY ES  TU SANTO":END
80 PRINT "HOLA ";N$;" ¿COMO ESTAS?"
```

Hay que tener en cuenta que el nombre hay que teclearlo en mayúsculas, ya que la cadena «Joaquín» no es igual que «JOAQUIN» ni la cadena «Ana» es igual a «ANA».

Por otra parte, y al igual que en el programa 2, en este programa también podríamos poner la condición compuesta contraria. Sólo tendríamos que cambiar las líneas 70 y 80 por las siguientes:

70 IF N$<>"JOAQUIN" AND N$<>"ANA" THEN PRINT "HOLA"; N$; "¿COMO ESTAS": END
80 PRINT "FELICIDADES"; N$, " , YA SE QUE HOY ES TU SANTO"

o por una única línea 70 utilizando la instrucción ELSE.

Finalmente, el programa 4 utiliza varias condiciones compuestas. El ordenador, tras pedir una nota (numérica), imprime la calificación correspondiente atendiendo al siguiente criterio:

NOTA < 5 .... SUSPENSO
5 <= NOTA < 7 .... APROBADO
7 <= NOTA < 9 .... NOTABLE
NOTA >=9 .......SOBRESALIENTE

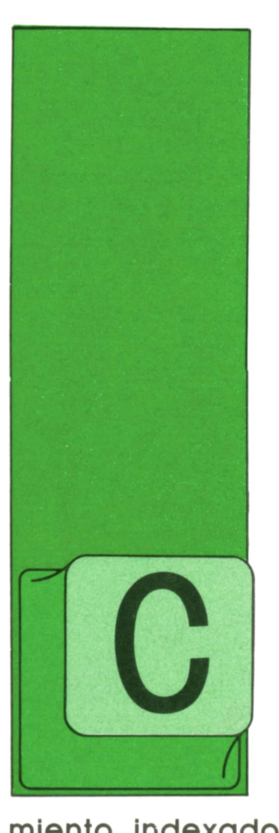
Además, si tecleamos una nota menor que cero o mayor que 10, el ordenador nos advertirá que dicha nota no es válida.

```
10 REM ********************
20 REM *  CALIFICACIONES  *
30 REM ********************
40 CLS
50 INPUT "TECLEE LA NOTA ";N
60 PRINT :PRINT
70 IF N<0 OR N>10 THEN PRINT "ESA NOTA
   NO ES VALIDA":END
80 IF N<5 THEN PRINT "SUSPENSO"
90 IF N>=5 AND N<7 THEN PRINT "APROBADO"
100 IF N>=7 AND N<9 THEN PRINT "NOTABLE"
110 IF N>=9 THEN PRINT "SOBRESALIENTE"
```

Por último, hay que señalar que todos los ordenadores no disponen de todos los operadores lógicos mencionados. Los más comunes son NOT, AND y OR. En cualquier caso, en la tabla de la figura 4 podemos ver operadores disponibles en cada una de las máquinas en estudio.

	NOT	AND	OR	XOR	EQU	IMP
Amstrad	X	X	X	X	X	X
Commodore	X	X	X			
IBM	X	X	X	X	X	X
MSX	X	X	X	X	X	X
Spectrum	X	X	X			


Operadores disponibles en los distintos ordenadores.



MAQUINA 6502

(COMMODORE 64) MODOS DE DIRECCIONAMIENTO

Direccionamiento indexado

CUANDO el dato se encuentra sumando al valor de la posición de memoria indicada el contenido del registro X o del registro Y, estamos utilizando el "direccionamiento indexado" o direccionamiento por índices.

Supongamos que hemos cargado el registro X con el número $10 mediante la instrucción ya vista LDX#$10. Si ahora introducimos el comando LDA $C000,X, el microprocesador interpretará que se debe cargar al ACU el valor que se encuentre en la posición de memoria $C000 + $10 = $C010.

De la misma manera podríamos haber cargado el registro Y con un número cualquiera, y después introducir el comando LDA $C000,Y.

Como puede observarse, se trata de un comando de tres bytes, uno para el código del comando LDA y dos para la posición de memoria especificada.

Este tipo de direccionamiento es especialmente útil en la programación de bucles. Adjudicando diferentes valores de X (incrementando o decreciendo el valor contenido en él), se pueden ir obteniendo al ser cargados en el ACU los valores que se encuentran en posiciones de memoria consecutivas como ya veremos.

5. También dentro de lo que hemos llamado direccionamiento "Zero page" existe el "direccionamiento indexado". Un ejemplo sería:

```
LDA $ 2B,X Comando de 2 Bytes
```

En este caso sólo *se utiliza el registro X* como índice, de tal forma que el comando

```
LDA $2B,Y
```

no tiene sentido alguno.

El correspondiente comando BASIC para el direccionamiento indexado sería:

```
A = PEEK ($C000 + X)
```

6. Veamos ahora un modo de direccionamiento un poco más complejo pero que permite una programación más potente. Es el "direccionamiento indirecto por índices".

— Por el registro Y: Ahora es el momento de explicar un poco eso de "Página Cero" y qué es lo que la hace tan importante.

Como dijimos, se trata de los 256 primeros bytes del área direccionable del procesador.

Allí se encuentran la mayor parte de los datos o variables necesarios para el correcto funcionamiento del ordenador, de tal manera que algunos registros deben tener un número concreto, mientras que otros pueden ser modificados de una manera útil.

¿Por qué se almacenan aquí? Sencillamente porque pueden ser direccionados con un solo byte.

Además, muchos de estos datos se encuentran en forma de "vectores, apuntadores o pointers".

Un vector se compone de dos posiciones contiguas de memoria en la página cero, cuyos contenidos apuntan a otra posición de memoria.

Veámoslo con un ejemplo: el área BASIC comienza en la posición de memoria 2049=$0801. Este valor debe ser "sabido" por el ordenador; de lo contrario, no podría colocar nuestros programas.

Pues bien, en las posiciones de memoria 43/44 se encuentran los valores 1 y 8, respectivamente.

PEEK (43) = 1 = $01
PEEK (44) = 8 = $08

Esto es realmente el byte bajo y el byte alto de la posición de memoria $ *08 01* = 2049.

Para hacer el cálculo en el sistema decimal basta con hacer:

Posición de memoria = byte bajo + 256 x byte alto

2049 = 1 + 256 x 8

Si cambiamos alguno o los dos valores de las posiciones 43/44, habremos cambiado el área BASIC disponible.

Así, la página cero dispone de vectores que le indican al microprocesador cómo se encuentra estructurada el área direccionable del ordenador.

Ahora podemos continuar con el direccionamiento indirecto mediante el registro Y.

Supongamos que la dirección de memoria 172 = $AC contiene un 60 = $3C y la 173 = $AD un 3 = $03.

Estas dos posiciones forman un apuntador hacia la posición 828 = $033C, que es donde empieza el buffer o memoria intermedia del casete.

La instrucción correspondiente al direccionamiento indirecto sería:

LDA ($AC),Y

y quiere decir "carga al acumulador lo que haya en la posición" $033C+Y, siendo $03 el número que hay en $AC y $3C el número que hay en $AD.

La correspondiente función BASIC sería:

A = PEEK (PEEK($AC) + 256 * PEEK($AD) + Y)

Alguno puede preguntarse: muy bien, pero ¿por qué no nos referimos directamente a la posición de memoria en cuestión?

LDA $XXXX direccionamiento absoluto o bien
LDA $XXXX,Y direccionamiento indexado en Y.

Si uno se fija bien, puede observar que con el direccionamiento indirecto con el registro Y no se necesita saber dónde se encuentra tal posición de memoria, tan sólo que ocupa el lugar Nº "Y" a partir de donde apunte el vector, y éste es inamovible.

Más claro: el vector que apunta al área del buffer del casete siempre está en las posiciones 172/173 y podemos leerlo o escribirlo (modificarlo) ahí. Sin embargo, dicha área puede ser trasladada a otra parte de la memoria, que no sea su configuración normal desde la posición 828-1023.

Ahora para referirnos al dato Nº "Y" de esa "tabla" o área, mediante el direccionamiento absoluto, o el indexado necesitamos conocer dónde ha sido trasladado el buffer del cassette, mientras que utilizando el direccionamiento indirecto, como el vector no puede ser modificado a otras posiciones que la 172/173, bastaría con escribir

LDA ($AC),Y

El microprocesador se encargará de saber dónde está ahora esa zona reservada al buffer del casete y cargar en el ACU el dato Nº "Y" dentro de ella.

— Por el registro X: también aquí dos direcciones contiguas de la página cero forman un apuntador a la dirección buscada.

La diferencia con el caso anterior es que en este caso suma primero el contenido del registro X a la posición de la página cero direccionada, formándose un nuevo apuntador.

La instrucción correspondiente a este direccionamiento es:

LDA ($BB,X)

y quiere decir: "Sitúate en la posición de memoria $BB+X y carga al ACU el contenido de la posición de memoria a la que apunta el vector $BB + X/$BC + X de la página cero".

Supongamos que hemos cargado el registro X con el número $00 mediante la instrucción LDX ı $00 y que a continuación introducimos el comando LDA ($BB,X).

En este caso, el apuntador formado será $BB + $00 / $BC + $00, o sea, $BB/$BC, que en decimal equivale a 187/188.

Este vector apunta siempre a la zona de memoria donde encontraremos el nombre del programa cargado en memoria.

En "condiciones normales", el valor en $BB es $F0 y el valor en $BC es $9F. Por tanto, el vector apunta a la dirección $9FF0 o, lo que es lo mismo, 240 + 256 * 159 = 40944 en decimal.

Pues bien, con el comando LDA ($BB,X), si en X hay un $00, se cargará en el ACU el valor de la posición $9FF0 = 40944 y corresponderá a la primera letra del nombre del fichero de 16 caracteres.

La instrucción equivalente en BASIC sería:

```
A = PEEK (PEEK ($BB + X) + 256 * PEEK ($BC + X))
```

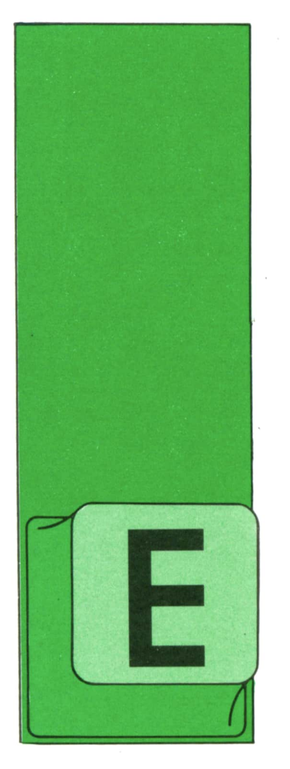
A continuación se expone una lista resumen de los comandos de carga:

Modo de direccionamiento	LDA	LDX	LDY
Inmediato	$A9	$A2	$A0
Absoluto	$AD	$AE	$AC
Zeropage	$A5	$A6	$A4
Absoluto indexado por X	$BD	—	$BC
Absoluto indexado por Y	$B9	$BE	—
Zeropage indexado por X	$B5	—	$B4
Zeropage indexado por Y	—	$B6	—
Indirecto indexado por Y	$B1	—	—
Indirecto indexado por X	$A1	—	—

Existen otros modos de direccionamiento que no tienen sentido dentro de los comandos de carga y que serán tratados más adelante.

PROGRAMAS

EDUCATIVOS • DE UTILIDAD • DE GESTION • DE JUEGOS



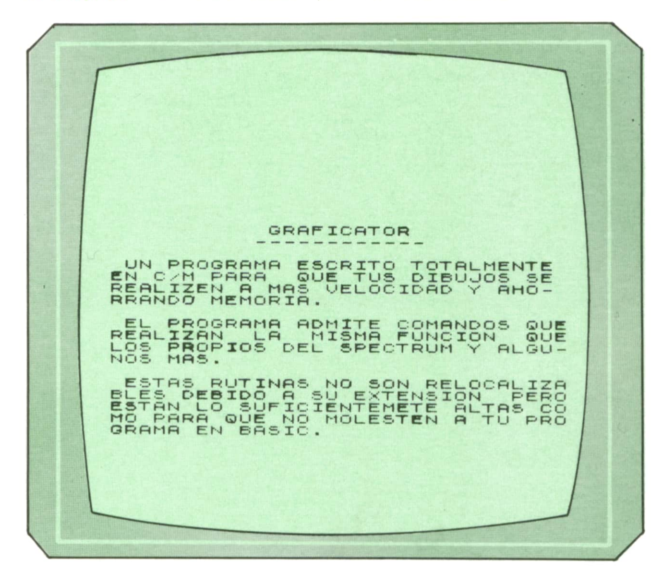


Programa: Graficator

EL primer programa de este tomo nos permitirá realizar gráficos más de prisa y ahorrando más memoria que en BASIC. Esto es gracias a una rutina en código máquina que nos permitirá tener todos los comandos gráficos del SPECTRUM y algunos más. Estos son:

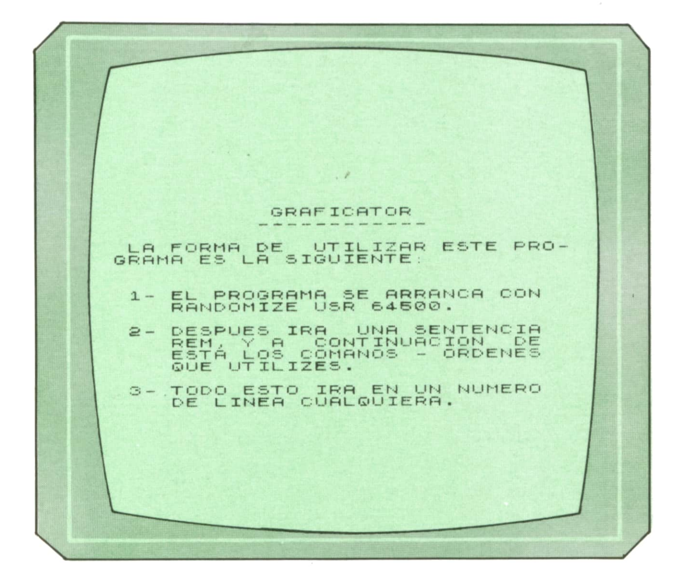


- P x,y . Coloca un punto en la pantalla.
- M x,y . Dibuja una línea.
- U y . Dibuja una línea hacia arriba.
- D y . Dibuja una línea hacia abajo.
- R x . Dibuja una línea hacia la derecha.
- L x . Dibuja una línea hacia la izquierda.
- I N . Cambia el color de la tinta.
- O n . Realiza la función over.
- C x,y,x . Nos dibuja un círculo.





Instrucciones del Graficator.



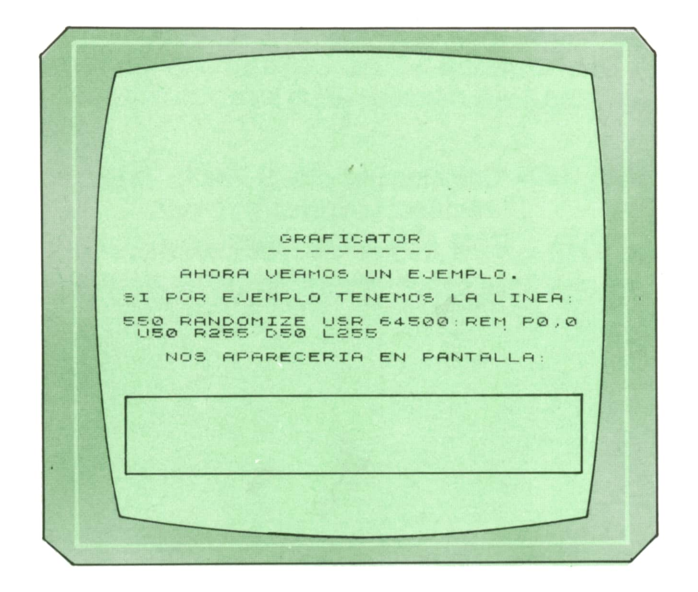


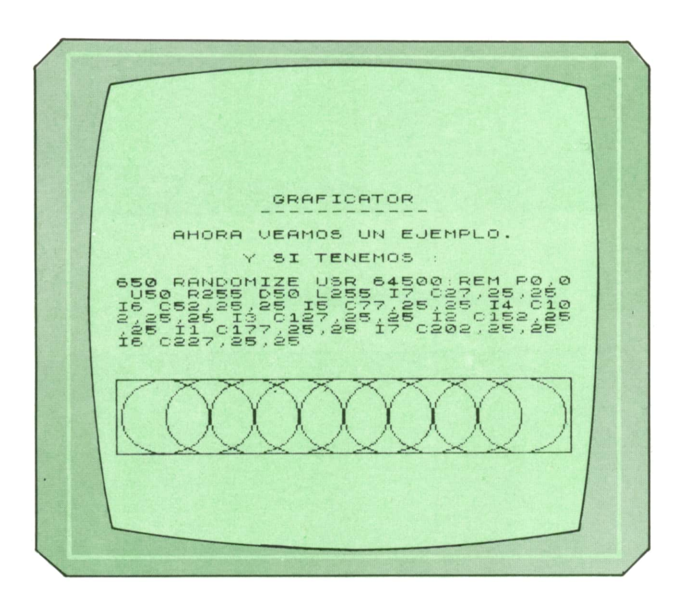
Instrucciones del Graficator.

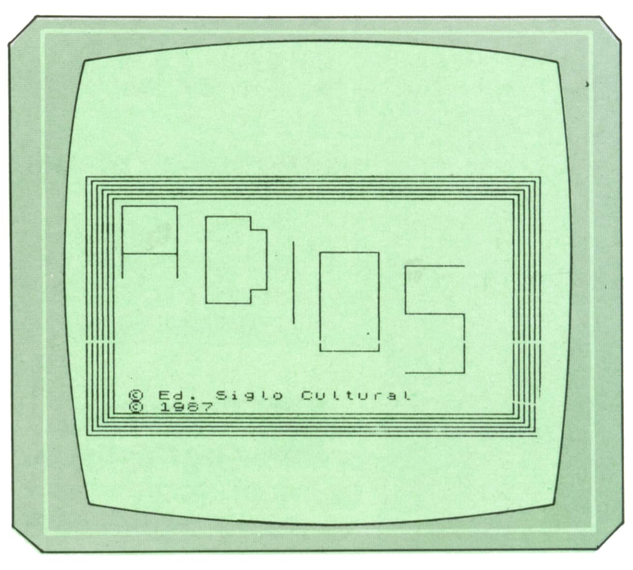
Todos estos comandos han de ir después de la instrucción:

RANDOMIZE USR 64500

en una línea REM.









Despedida del programa.



El programa nos muestra un ejemplo de su funcionamiento.



Otro ejemplo de su funcionamiento.

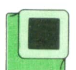
La utilización del programa no reviste ninguna dificultad, ya que el programa cargador no sólo introduce el código máquina en memoria, sino que nos explica cómo utilizarlo, nos muestra algunos ejemplos y nos permite grabarlo en forma de BYTES en la cinta.

```
  10>PRINT #0;AT 1,0;FLASH 1;PAP
ER 7;INK 1;INVERSE 1;"CARGANDO D
ATAS.ESPERA UN MOMENTO"
  20 PRINT AT 12,8; INK 5;"CHECK
 SUM = 0": GO SUB 9500
  30 BEEP .5,0: BEEP .5,5: BEEP
.5,8
  40 CLS : LET Y=0: LET X=11: LE
T A$="GRAFICATOR": LET C=6: GO S
```

```
UB 8000: LET Y=1: LET X=10: LET
A$="------------": LET C=3: GO S
UB 8000
  50 LET A$=" UN PROGRAMA ESCRIT
O TOTALMENTE EN C/M PARA  QUE TU
S DIBUJOS SE REALIZEN A MAS VELO
CIDAD Y AHO- RRANDO MEMORIA.": L
ET X=0: LET Y=3: LET C=4: GO SUB
 8000
```

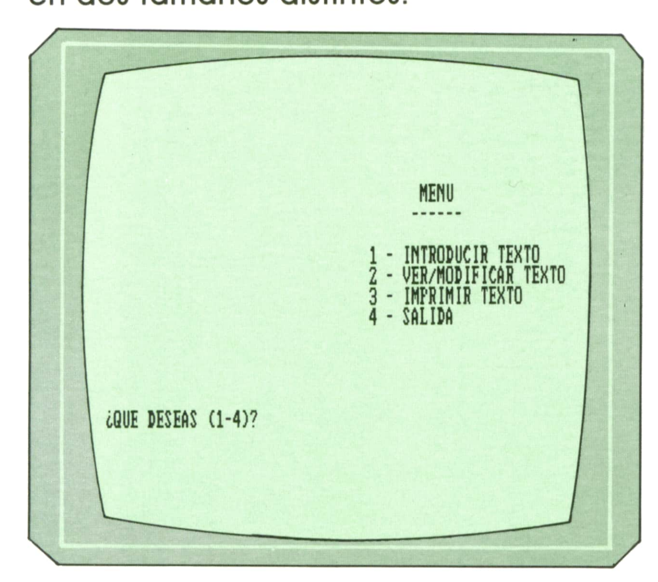
```
  60 LET A$=" EL PROGRAMA ADMITE
 COMANDOS QUEREALIZAN  LA  MISMA
 FUNCION  QUELOS PROPIOS DEL SPE
CTRUM Y ALGU-NOS MAS.": LET X=0:
 LET Y=8: GO SUB 8000
  70 LET A$=" ESTAS RUTINAS NO S
ON RELOCALIZABLES DEBIDO A SU EX
TENSION  PEROESTAN LO SUFICIENTE
METE ALTAS COMO PARA QUE NO MÓLE
STEN A TU PROGRAMA EN BASIC.": L
ET X=0: LET Y=13: GO SUB 8000
  90 GO SUB 7000
 100 INK 6: CLS : LET A$="GRAFIC
ATOR": LET X=11: LET Y=0: LET C=
4: GO SUB 8000: LET A$="--------
----": LET X=10: LET Y=1: LET C=
5: GO SUB 8000
 110 LET A$=" LA FORMA DE  UTILI
ZAR ESTE PRO-GRAMA ES LA SIGUIEN
TE:": LET X=0: LET Y=3: LET C=6:
 GO SUB 8000
 120 LET A$="1- EL PROGRAMA SE A
RRANCA CON       RANDOMIZE USR 64
500.": LET X=1: LET Y=7: LET C=7
: GO SUB 8000
 130 LET A$="2- DESPUES IRA  UNA
 SENTENCIA       REM, Y A  CONTIN
UACION  DE       ESTA LOS COMANOS
 - ORDENES       QUE UTILIZES.":
LET X=1: LET Y=10: GO SUB 8000
 140 LET A$="3- TODO ESTO IRA EN
 UN NUMERO       DE LINEA CUALQUI
ERA.": LET X=1: LET Y=15: GO SUB
 8000: GO SUB 7000
 200 CLS : LET A$="GRAFICATOR":
LET Y=0: LET X=11: LET C=6: GO S
UB 8000: LET A$="------------":
LET X=10: LET Y=1: LET C=4: GO S
UB 8000
 210 LET A$=" LOS COMANDOS POSIB
LES SON:": LET X=2: LET Y=3: LET
 C=7: GO SUB 8000
 220 LET A$="Px,y --- PLOT EN LA
 POSICION                x,y. IGUAL
 QUE EN EL               BASIC.": L
ET X=2: LET Y=5: LET C=6: GO SUB
 8000
 230 LET A$="Mx,y --- TRAZA UNA
LINEA HAS                TA LA POSI
CION  x,y                RELATIVA A
L  ULTIMO                PLOT. COMO
  DRAW EN                BASIC.": L
ET X=2: LET Y=9: GO SUB 8000
 240 LET A$="On ----- OVER n. n
PUEDE TE-                NER CUALQU
IER VALOR                ENTRE 0 Y
255. SI n                ES 0 SE HA
CE OVER 0                Y CUALQUIE
R OTRO VA                LOR OVER 1
.COMO  EN                BASIC.": L
ET X=2: LET Y=15: GO SUB 8000
 250 GO SUB 7000: GO SUB 7500
 300 LET A$="In ----- INK n. DON
DE n SERA                UN NUMERO
ENTRE 0 Y                7. IGUAL Q
UE EN BA-                SIC.": LET
 X=2: LET Y=5: GO SUB 8000
 310 LET A$="Cx,y,r - CIRCLE x,y
,r.  DON-                DE x E y
SON   LAS                COORDENADA
S DEL CEN                TRO DEL CI
RCULO Y r                EL RADIO D
EL MISMO.                COMO EN BA
SIC.": LET X=2: LET Y=10: GO SUB
 8000
 320 GO SUB 7000: GO SUB 7500
 400 LET A$="Un ----- UP n. DIBU
JA UNA LI                NEA DE  LO
NGITUD  n                HACIA ARRI
BA.": LET X=2: LET Y=5: GO SUB 8
000
 410 LET A$="Dn ----- DOWN n. DI
BUJA  UNA                LINEA DE L
ONGITUD n                HACIA ABAJ
O.": LET X=2: LET Y=9: GO SUB 80
00
 420 LET A$="Rn ----- RIGHT n.DI
BUJA  UNA                LINEA HACI
A LA DERE                CHA DE LON
GITUD n.": LET X=2: LET Y=13: GO
 SUB 8000
 430 LET A$="Ln ----- LEFT n. DI
BUJA  UNA                LINEA DE L
ONGITUD n                HACIA LA I
ZQUIERDA.": LET X=2: LET Y=17: G
O SUB 8000
 440 LET A$="LOS ESPACIOS EN BLA
NCO NO CAUSANNINGUN EFECTO.": LE
T X=0: LET Y=20: LET C=5: GO SUB
 8000: GO SUB 7000
 500 CLS : LET A$="GRAFICATOR":
LET C=7: LET X=11: LET Y=0: GO S
UB 8000: LET A$="------------":
LET C=3: LET X=10: LET Y=1: GO S
UB 8000
 510 LET A$="AHORA VEAMOS UN EJE
MPLO.": LET X=4: LET Y=3: LET C=
6: GO SUB 8000
 520 LET A$="SI POR EJEMPLO TENE
MOS LA LINEA:": LET X=0: LET C=4
: LET Y=5: GO SUB 8000
 530 LET A$="550 RANDOMIZE USR 6
4500:REM P0,0 U50 R255 D50 L255"
: LET X=0: LET Y=7: LET C=7: GO
SUB 8000
 540 LET A$="NOS APARECERIA EN P
ANTALLA:": LET X=3: LET Y=10: LE
T C=6: GO SUB 8000
 550 RANDOMIZE USR 64500: REM P0
,0 U50 R255 D50 L255
 560 GO SUB 7000: GO SUB 7500
 600 LET A$="Y SI TENEMOS :": LE
T X=9: LET Y=5: GO SUB 8000
 610 LET A$="650 RANDOMIZE USR 6
4500:REM P0,0 U50 R255 D50 L255
I7 C27,25,25 I6 C52,25,25 I5 C77
,25,25 I4 C102,25,25 I3 C127,25,
25 I2 C152,25,25 I1 C177,25,25 I
7 C202,25,25 I6 C227,25,25": LET
 X=0: LET Y=7: LET C=7: GO SUB 8
000
 650  RANDOMIZE  USR  64500: REM
  P0,0 U50 R255 D50 L255 I7 C27,
25,25 I6 C52,25,25 I5 C77,25,25
I4 C102,25,25 I3 C127,25,25 I2 C
152,25,25 I1 C177,25,25 I7 C202,
25,25 I6 C227,25,25
```

```
660 GO SUB 7000: GO SUB 7500
700 CLS : PRINT AT 10,6; INK 6;
INVERSE 1;" GRABO EL PROGRAMA?
"
710 IF INKEY$="N" OR INKEY$="n"
THEN  BEEP .1,35: GO TO 800
720 IF INKEY$<>"S" AND INKEY$<>
"s" THEN  GO TO 710
730 BEEP .01,20: PRINT AT 21,11
; INK 5;"PULSA ENTER": IF INKEY$
<>CHR$ 13 THEN  GO TO 730
740 SAVE "GRAFICATOR" LINE 1
800 FOR I=1 TO 100: NEXT I: CLS
: PRINT AT 10,3; INK 4; INVERSE
1;" GRABO EL CODIGO MAQUINA? ":
810 IF INKEY$="N" OR INKEY$="n"
THEN  BEEP .1,35: GO TO 9600
820 IF INKEY$<>"S" AND INKEY$<>
"s" THEN  GO TO 810
830 BEEP .1,20: PRINT AT 21,11;
INK 7;"PULSA ENTER": IF INKEY$<
>CHR$ 13 THEN  GO TO 830
840 SAVE "GRAF-CODE "CODE 64500
,500: GO TO 9600
6999 STOP
7000 FOR I=0 TO 1 STEP 0: FOR J=
0 TO 7: PRINT #0;AT 1,8; INK J;"
PULSA UNA TECLA": BEEP .01,J+20:
 IF INKEY$<>"" THEN  LET J=8: LE
T I=2
7010 NEXT J: NEXT I: RETURN
7500 FOR I=5 TO 21: PRINT AT I,0
;"
   ": NEXT I: PRINT #0;AT 1,8;"
                ": RETURN
8000 PRINT AT Y,X;: FOR I=1 TO L
EN A$: PRINT  INK C;A$(I);: BEEP
 .01,35: NEXT I: BEEP .05,-10: P
AUSE 20: RETURN
9000 DATA 217,229,217,42,93,92,3
5,126,254,234,32,55,35,126,254,3
2,40,250,254,13,40,47,254,77,202
,138,252,254,80,202,107
9010 DATA 252,254,85,202,118,253
,254,68,202,144,253,254,76,202,1
71,253,254,82,202,198,253,254,67
,202,221,252,254,73,40,30,254
9020 DATA 79,40,8,24,0,207,9,217
,225,217,201,35,126,254,49,40,6,
253,54,87,0,24,183,253,54,87,3,2
4,177,35,126
9030 DATA 205,210,252,56,225,237
,91,141,92,214,48,203,131,203,13
9,203,147,131,50,141,92,50,143,9
2,24,149,205,15,253,126,254
9040 DATA 44,32,196,58,4,91,95,2
13,205,15,253,58,4,91,209,87,235
,213,205,49,253,225,43,195,0,252
,35,17,0,0,126
9050 DATA 254,45,40,5,22,1,43,24
,2,22,255,213,205,15,253,209,126
,254,44,32,147,58,4,91,79,35,126
,254,45,40,5
9060 DATA 30,1,43,24,2,30,255,21
3,205,15,253,58,4,91,209,71,254,
0,32,1,90,121,254,0,32,1,83,229,
205,186,36
9070 DATA 225,43,195,0,252,254,5
8,48,5,254,48,56,1,201,55,201,20
5,15,253,254,44,194,55,252,58,4,
91,229,205,40,45
9080 DATA 225,205,15,253,254,44,
194,55,252,58,4,91,229,205,40,45
,225,205,15,253,58,4,91,229,205,
40,45,205,45,35,225
9090 DATA 43,195,0,252,17,0,0,23
7,83,4,91,35,126,205,210,252,216
,214,48,229,42,4,91,203,37,93,20
3,37,203,37,133
9100 DATA 131,50,4,91,225,24,229
,34,125,92,62,175,148,103,229,23
0,7,198,64,79,124,31,31,31,230,3
1,71,230,24,87,124
9110 DATA 230,192,95,97,125,31,3
1,31,230,31,111,123,128,146,95,2
2,0,229,213,225,41,41,41,41,41,2
09,25,209,123,230,7
9120 DATA 71,62,8,144,71,62,1,13
5,16,253,31,182,119,201,205,15,2
53,58,4,91,71,229,42,125,92,36,2
29,197,205,49,253
9130 DATA 193,225,16,246,225,43,
195,0,252,205,15,253,58,4,91,71,
229,42,125,92,37,229,197,205,49,
253,193,225,5,32,245
9140 DATA 225,43,195,0,252,205,1
5,253,58,4,91,71,229,42,125,92,4
5,229,197,205,49,253,193,225,5,3
2,245,225,43,195,0
9150 DATA 252,205,15,253,58,4,91
,71,229,42,125,92,44,229,197,205
,49,253,193,225,16,246,225,43,19
5,0,252,0,0,0,0
9500 LET CHECK=0: FOR I=64500 TO
 64995: READ A: POKE I,A: LET CH
ECK=CHECK+A: PRINT AT 12,20; INK
 5;CHECK: NEXT I
9510 IF CHECK<>60804 THEN  CLS :
 PRINT AT 9,7; INVERSE 1; INK 6;
" ERROR EN DATAS ";AT 11,5;" INT
ENTALO DE NUEVO ": GO TO 9999
9520 RETURN
9600 REM  DESPEDIDA
9610 CLS : RANDOMIZE USR 64500:
REM I3 P0,0 M0,175 M255,0 M0,-17
5 M-255,0 P3,3 M0,169 M249,0 M0,
-169 M-249,0 P6,6 M0,163 M243,0
M0,-163 M-243,0 I6 P9,9 M0,157 M
237,0 M0,-157 M-237,0 P12,12 M0,
151 M231,0M0,-151 M-231,0 P15,15
 M0,145 M225,0 M0,-145 M-225,0
9620 PAPER 5: RANDOMIZE USR 6450
0: REM I0 P20,107 M0,48 M31,0 M0
,-48 P20,123 M31,0 P67,91 M0,57
M25,0 M0,-9 M9,0 M0,-40 M-9,0 M0
,-9 M-25,0 P116,76 M0,55
9630 RANDOMIZE USR 64500: REM P1
31,59 M0,65 M33,0 M0,-65 M-33,0
P179,43 M33,0 M0,40 M-33,0 M0,32
 M33,0
9640 PAPER 0: INK 7: PRINT AT 17
,3; Fco. Morales.";AT 18,3;"c/
 Corazon de maria, 13";AT 19,3;"
28002 MADRID": PAUSE 0
```



Programa: Diario

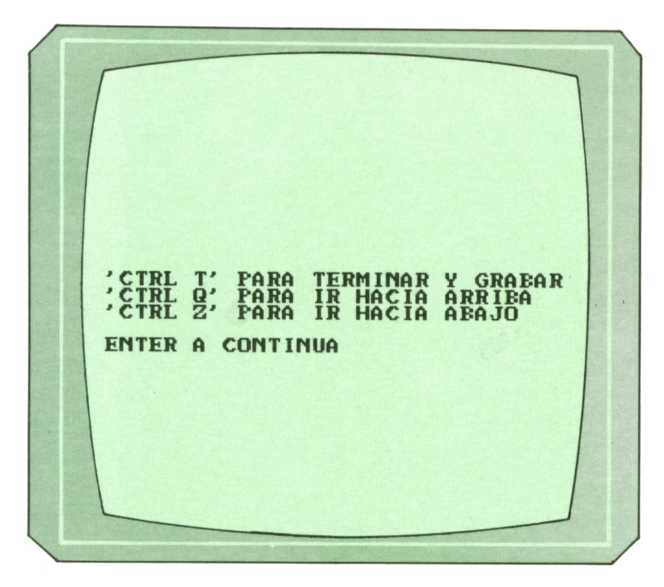
El programa que proponemos a continuación nos permitirá tener nuestro propio diario electrónico con todas las facilidades que eso conlleva. Gracias a que el diario es electrónico, podremos modificar en cualquier momento cualquier parte del texto, visualizarlo e imprimirlo en dos tamaños distintos.





Pantalla de menú del Programa Diario.

El programa está organizado como si de un editor de textos se tratase. La única diferencia es que es menos potente, pero no por ello menos útil.





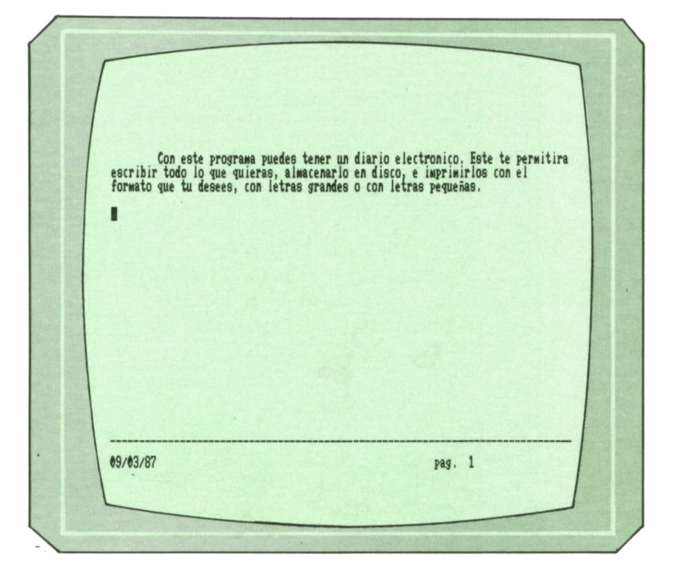
Aparte de comandos normales que puedes utilizar, también puedes hacer uso de éstos.

Una vez introducido el programa en memoria nos aparece un menú con cuatro opciones. Estas son:

— Introducir textos.
— Ver/modificar texto.
— Imprimir texto.
— Salir del programa.

En cualquiera de las tres primeras opciones que elijamos, el ordenador nos preguntará el nombre del fichero que contiene nuestro diario y la fecha. La fecha nos la pide porque el diario, al igual que un diario de verdad, está organizado por días.

A la hora de ponernos a escribir no hay ningún problema en escribir una fecha anterior a la última escrita. El ordenador la gestionará adecuadamente y la ordenará.





Ejemplo de ejecución del Programa Diario.

Las teclas que podemos utilizar para escribir el texto son las siguientes:

— Las teclas del cursor.
— La tecla TAB.
— La tecla BACKSPACE (borra el carácter de la izquierda).
— CONTROL Q para ir a la página anterior.
— CONTROL Z para ir a la página siguiente.
— CONTROL T para terminar la sesión.

Lo único que falta por decir es que por cada fecha contamos con tres páginas

distintas de texto que podemos llenar o no. Para pasar de una página a otra podemos utilizar las teclas CONTROL Q y CONTROL Z o bien poner el cursor al final de una página para ir a la siguiente o bien poner el cursor al principio de una página para ir a la anterior.

Aunque este programa sólo es válido para el IBM, en tomos sucesivos veremos las distintas versiones de este programa para otros ordenadores.

```
100 REM *********************************
110 REM *                               *
120 REM *           DIARIO              *
130 REM *                               *
140 REM *  POR: PETER BERGMANN          *
150 REM *                               *
160 REM *********************************
170 REM *                               *
180 REM *          VARIABLES            *
190 REM *          ---------            *
200 REM *                               *
210 REM * T$     ARRAY DE PAGINAS Y FILAS*
220 REM * F$     NOMBRE DEL FICHERO     *
230 REM * E$     ARRAY TEMPORAL         *
240 REM * X$     FLAG       (S/N)       *
250 REM * I,J    CONTADORES             *
260 REM * O      OPCION DEL MENU        *
270 REM * L$     FICHERO EXISTENTE FLAG *
280 REM * S$     STRING TEMPORAL        *
290 REM * C$     RESPUESTA PARA CONT.   *
300 REM * D      FECHA                  *
310 REM * DI     FECHA (DEL REGISTRO)   *
320 REM * P      NUMERO DE LA PAGINA    *
330 REM * R      NUMERO DE LA FILA      *
340 REM * C      NUMERO DE LA COLUMNA   *
350 REM * RL     NUMERO DE LA FILA ANTES *
360 REM * D$     FECHA                  *
370 REM * M$,M   MES                    *
380 REM * W$,W   DIA                    *
390 REM * Y$,Y   A%O                    *
400 REM * R$     RESPUESTA (S/N)        *
410 REM * A$     INPUT CHARACTER        *
420 REM * S      NUMERO DE DIAS EN MES  *
430 REM * G$     TAMA%O  (G/P)          *
440 REM *                               *
450 REM *********************************
460 REM
470 REM *** PRESENTACION ***
480 REM
490 KEY OFF
500 CLS
510 PRINT STRING$(79,"*")
520 FOR I = 1 TO 19
530    PRINT  "*"; TAB(79); "*"
540 NEXT I
550 PRINT STRING$(79,"*")
560 LOCATE 8,36: PRINT "DIARIO"
570 LOCATE 9,35: PRINT "--------"
580 LOCATE 12,26: PRINT "(c) Ed. Siglo Cultural, 1987"
590 LOCATE 24,1
600 FOR I = 1 TO 1200: NEXT I
610 REM
620 REM *** DECLARACION DE ARRAYS ***
630 REM
640 DIM E$(80)
650 DIM T$(3,20)
660 REM
```

```
670 REM ***************************************
680 REM *                MENU                 *
690 REM ***************************************
700 REM
710 CLS
720 LOCATE 3,38: PRINT "MENU"
730 LOCATE 4,37: PRINT "------"
740 LOCATE 6,32: PRINT "1 - INTRODUCIR TEXTO"
750 LOCATE 7,32: PRINT "2 - VER/MODIFICAR TEXTO"
760 LOCATE 8,32: PRINT "3 - IMPRIMIR TEXTO"
770 LOCATE 9,32: PRINT "4 - SALIDA"
780 LOCATE 14,1: PRINT "(QUE DESEAS (1-4)";
790 INPUT O
800 IF (O < 1) OR (O > 4) GOTO 1730
810 ON O GOSUB 940,1080,1260,830
820 GOTO 660
830 REM
840 REM *** SALIDA ***
850 REM
860 CLS
870 PRINT "ADIOS..."
880 FOR I=1 TO 10
890    PRINT
900 NEXT I
910 KEY ON
920 END
930 GOTO 920
940 REM
950 REM **************************************
960 REM *       ENTRADA DEL TEXTO            *
970 REM **************************************
980 REM
990 CLS
1000 GOSUB 1400
1010 GOSUB 1500
1020 GOSUB 2710
1030 GOSUB 1830
1040 IF X$ = "S" THEN GOSUB 2150: GOTO 1070
1050 GOSUB 3470
1060 GOSUB 2220
1070 RETURN
1080 REM
1090 REM ***********************************
1100 REM *      BUSCAR/CAMBIAR TEXTO       *
1110 REM ***********************************
1120 REM
1130 CLS
1140 GOSUB 1400
1150 GOSUB 1500
1160 GOSUB 2710
1170 GOSUB 1830
1180 IF X$ = "N" THEN GOSUB 2080: GOTO 1240
1190 LET P = 1
1200 GOSUB 2570
1210 GOSUB 3470
1220 GOSUB 2360
1230 GOSUB 2220
1240 RETURN
1250 REM
1260 REM ***********************************
1270 REM *        IMPRIMIR TEXTO           *
1280 REM ***********************************
1290 REM
1300 CLS
1310 GOSUB 1400
1320 GOSUB 1500
1330 GOSUB 2710
1340 GOSUB 1830
1350 IF X$ = "N" THEN GOSUB 2080: GOTO 1390
```

```
1360 LET P = 1
1370 GOSUB 3330
1380 GOSUB 3200
1390 RETURN
1400 REM
1410 REM *** INICIALIZACION DE ARRAYS ***
1420 REM
1430 FOR I = 1 TO 3
1440   FOR J = 1 TO 20
1450       LET T$(I,J) = SPACE$(80)
1460       LET E$(J) = " "
1470   NEXT J
1480 NEXT I
1490 RETURN
1500 REM
1510 REM *** NOMBRE DEL FICHERO A USAR ***
1520 REM
1530 CLS
1540 LET L$ = "S"
1550 LOCATE 1,34
1560 IF O = 1 THEN PRINT "FICHERO DE SALIDA" ELSE PRINT "FICHERO DE ENTRADA"
1570 LOCATE 2,33
1580 IF O = 1 THEN PRINT "-----------------" ELSE PRINT "------------------"
1590 LOCATE 4,1: PRINT "(CUAL ES EL NOMBRE DEL FICHERO"
1600 INPUT F$
1610 LOCATE 6,1: PRINT "EL NOMBRE ES ";F$
1620 ON ERROR GOTO 1790
1630 OPEN F$ FOR INPUT AS #1
1640 CLOSE #1
1650 LOCATE 9,1
1660 IF L$ = "N" THEN PRINT "ES UN FICHERO NUEVO" ELSE PRINT "ESTE FICHERO YA EX
ISTE"
1670 IF (O <> 2) OR (L$ <> "N") THEN GOTO 1720
1680 LOCATE 11,1: PRINT "NECESITAS TENER UN FICHERO QUE EXISTA"
1690 PRINT "(ENTER PARA CONTINUA)"
1700 C$ = INPUT$(1)
1710 GOTO 1500
1720 LOCATE 11,1: PRINT "(ESTA CORRECTO (S/N)";
1730 INPUT R$
1740 IF R$ = "N" THEN GOTO 1500
1750 IF R$ <> "S" THEN GOTO 1720
1760 ON ERROR GOTO 0
1770 RETURN
1780 REM
1790 REM *** FILE ERROR ***
1800 REM
1810 LET L$ = "N"
1820 RESUME 1650
1830 REM
1840 REM *** BUSCAR/LEER FICHERO ANTIGUO ***
1850 REM
1860 LET X$ = "Y"
1870 IF L$ = "N" THEN GOTO 2070
1880 CLS
1890 PRINT "ESPERE, POR FAVOR...    CARGANDO REGISTROS"
1900 LET X$ = "N"
1910 OPEN F$ FOR INPUT AS #1
1920 OPEN "TEMP" FOR OUTPUT AS #2
1930 WHILE NOT EOF(1)
1940    INPUT #1, DI
1950    IF DI = D THEN LET X$ = "S" ELSE WRITE #2, DI
1960    FOR I = 1 TO 3
1970       FOR J = 1 TO 20
1980          INPUT #1, S$
1990          IF DI <> D THEN WRITE #2, S$: GOTO 2010
2000          IF S$ = " " THEN LET T$(I,J) = SPACE$(80) ELSE LET T$(I,J) = S$
2010       NEXT J
2020    NEXT I
2030 WEND
```

```
2040 CLOSE #1
2050 CLOSE #2
2060 IF X$ = "N" THEN KILL "TEMP"
2070 RETURN
2080 REM
2090 REM *** FECHA NO ENCONTRADA ***
2100 REM
2110 CLS
2120 LOCATE 10,10: PRINT M$;"/";W$;"/";Y$;" NO SE ENCONTRO EN EL FICHERO ";F$
2130 LOCATE 11,10: PRINT "(ENTER PARA CONTINUA)";: C$ = INPUT$(1)
2140 RETURN
2150 REM
2160 REM *** LA FECHA EXISTE ***
2170 REM
2180 CLS
2190 LOCATE 10,32: PRINT "ESTA FECHA EXISTE"
2200 LOCATE 20,32: PRINT "(ENTER PARA CONTINUA)";:C$ = INPUT$(1)
2210 RETURN
2220 REM
2230 REM *** GRABACION ***
2240 REM
2250 OPEN F$ FOR APPEND AS #1
2260 WRITE #1, D
2270 FOR I = 1 TO 3
2280    FOR J = 1 TO 20
2290       IF T$(I,J) > SPACE$(80) THEN WRITE #1, T$(I,J) ELSE WRITE #1, " "
2300    NEXT J
2310 NEXT I
2320 CLOSE #1
2330 LOCATE 10,10: PRINT "TEXTO DE ";M$;"/";W$;"/";Y$;" GRABADO EN ";F$
2340 LOCATE 12,10: PRINT "(ENTER PARA CONTINUA)";: C$ = INPUT$(1)
2350 RETURN
2360 REM
2370 REM ***  CAMBIAR FICHERO ANTIGUO ***
2380 REM
2390 CLS
2400 PRINT "ESPERE, POR FAVOR...   ESCRIBIENDO FICHERO"
2410 OPEN F$ FOR OUTPUT AS #1
2420 OPEN "TEMP" FOR INPUT AS #2
2430 IF EOF(2) THEN GOTO 2530
2440 INPUT #2, DI
2450 IF EOF(2) THEN GOTO 2530
2460 WRITE #1, DI
2470 FOR I = 1 TO 3
2480    FOR J = 1 TO 20
2490       INPUT #2, S$: WRITE #1, S$
2500    NEXT J
2510 NEXT I
2520 GOTO 2430
2530 CLOSE #1
2540 CLOSE #2
2550 KILL "TEMP"
2560 RETURN
2570 REM*
2580 REM *** IMPRIMIR PAGINA ***
2590 REM
2600 CLS
2610 LOCATE 1,1
2620 FOR R = 1 TO 20
2630    IF T$(P,R) = SPACE$(80) THEN PRINT: GOTO 2650
2640    LOCATE R,1: PRINT T$(P,R)
2650 NEXT R
2660 LOCATE 21,1: PRINT STRING$(80,"_")
2670 LOCATE 23,1: PRINT M$;"/";W$;"/";Y$,,,,"pag. ";P
2680 LOCATE 1,1
2690 LET R = 1: LET RL = 1
2700 RETURN
2710 REM
2720 REM *** LECTURA DE LA FECHA ***
```

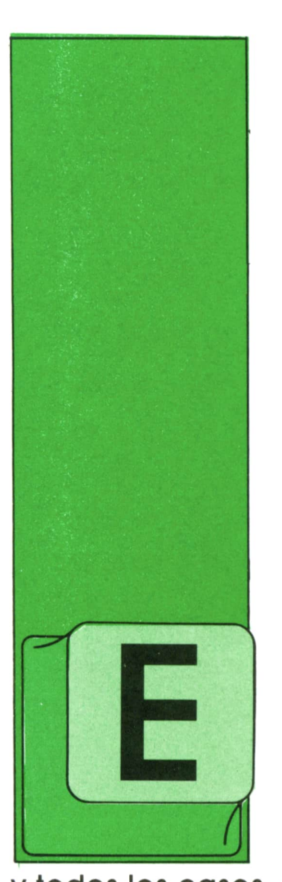
```
2730 REM
2740 CLS
2750 LOCATE 2,1: PRINT "INTRODUCE LA FECHA (MMDDYY): ";
2760 LET I = 1
2770 WHILE I < 7
2780    C$ = INPUT$(1)
2790    C = ASC(C$)
2800    IF (C$ = CHR$(8)) AND (I > 1) THEN PRINT CHR$(29);" ";CHR$(29);:I = I -
1: GOTO 2780
2810    IF (C < 48) OR (C > 57) THEN GOTO 2780
2820    PRINT C$;
2830    LET E$(I) = C$
2840    I = I + 1
2850 WEND
2860 LET D$ = ""
2870 FOR I = 1 TO 6
2880   D$ = D$ + E$(I)
2890 NEXT I
2900 GOSUB 3030
2910 IF X$ = "S" THEN GOTO 2950
2920 PRINT :PRINT "FECHA NO VALIDA"
2930 LOCATE 2,1: PRINT SPACE$(80)
2940 GOTO 2750
2950 CLS
2960 PRINT "LA FECHA ES ";M$;"/";W$;"/";Y$
2970 LOCATE 4,1: PRINT "(ES CORRECTA";
2980 INPUT R$
2990 IF R$ = "N" THEN GOTO 2710
3000 IF R$ <> "S" THEN GOTO 2970
3010 D = VAL(D$)
3020 RETURN
3030 REM
3040 REM ***  COMPROBAR FECHA ***
3050 REM
3060 LET X$ = "S"
3070 LET S = 30
3080 LET M$ = LEFT$(D$,2)
3090 LET W$ = MID$(D$,3,2)
3100 LET Y$ = RIGHT$(D$,2)
3110 M = VAL(M$): W = VAL(W$): Y = VAL(Y$)
3120 PRINT :PRINT M$;"/";W$;"/";Y$
3130 IF (M < 1) OR (M > 12) THEN PRINT "MES NO INVALIDO": LET X$ = "N": GOTO 319
0
3140 IF (Y < 85) OR (Y > 90) THEN PRINT "A%O NO VALIDO": LET X$ = "N": GOTO 3190
3150 IF (M = 1) OR (M = 3) OR (M = 5) OR (M = 7) OR (M = 8) OR (M = 10) OR (M =
12) THEN LET S = 31
3160 IF M = 2 THEN LET S = 28
3170 IF (M = 2) AND (Y = 88) THEN LET S = 29
3180 IF (W < 1) OR (W > S) THEN PRINT "DIA NO VALIDO": LET X$ = "N"
3190 RETURN
3200 REM
3210 REM *** SALIDA POR IMPRESORA ***
3220 REM
3230 CLS
3240 PRINT "IMPRIMIENDO ..."
3250 FOR P = 1 TO 3
3260    LPRINT M$;"/";W$;"/";Y$,,,,"pag. ";P:LPRINT: LPRINT
3270    FOR R = 1 TO 20
3280       LPRINT T$(P,R)
3290    NEXT R
3300    LPRINT CHR$(12)
3310 NEXT P
3320 RETURN
3330 REM
3340 REM *** PREPARAR LA IMPRESORA ***
3350 REM
3360 LPRINT CHR$(27)+CHR$(64)
3370 CLS
3380 LOCATE 2,1: PRINT "(QUIERES IMPRESION GRANDE O PEQUE%A (G/P)";
```

```
3390 INPUT G$
3400 IF G$ = "P" THEN LPRINT CHR$(27)+CHR$(15)
3410 IF G$ = "G" THEN LPRINT CHR$(18)
3420 IF (G$ <> "P") AND (G$ <> "G") THEN GOTO 3370
3430 CLS
3440 LOCATE 2,1: PRINT "COLOCA EL PAPEL Y PULSA ENTER";
3450 C$ = INPUT$(1)
3460 RETURN
3470 REM
3480 REM *** PANTALLA DE ENTRADA ***
3490 REM
3500 IF O = 2 THEN GOTO 3580
3510 CLS
3520 LOCATE 2,1: PRINT "'CTRL T' PARA TERMINAR Y GRABAR"
3530 LOCATE 3,1: PRINT "'CTRL Q' PARA IR HACIA ARRIBA"
3540 LOCATE 4,1: PRINT "'CTRL Z' PARA IR HACIA ABAJO"
3550 LOCATE 6,1: PRINT "ENTER A CONTINUA"
3560 C$ = INPUT$(1)
3570 CLS
3580 LET P = 1
3590 LET RL = 1
3600 LOCATE 21,1: PRINT STRING$(80,"_")
3610 LOCATE 23,1: PRINT M$;"/";W$;"/";Y$,,,,"PAGE ";P
3620 LOCATE 1,1,1
3630 A$ = INKEY$: IF A$ = "" THEN GOTO 3630
3640 IF A$ = CHR$(8) THEN PRINT CHR$(29);" ";CHR$(29);
3650 IF A$ = CHR$(17) THEN GOSUB 4010: GOTO 3630
3660 IF A$ = CHR$(26) THEN GOSUB 4080: GOTO 3630: REM* GO FORWARD ONE PAGE
3670 IF LEN(A$) = 2 THEN GOSUB 3720: GOTO 3630
3680 IF A$ = CHR$(20) THEN GOTO 3710
3690 GOSUB 3850
3700 GOTO 3630
3710 RETURN
3720 REM
3730 REM *** MOVIMIENTO DEL CURSOR ***
3740 REM
3750 R = CSRLIN
3760 C = POS(0)
3770 IF MID$(A$,2,1)="H" AND (R > 1) THEN PRINT CHR$(30);
3780 IF MID$(A$,2,1)="P" AND (R < 20) THEN PRINT CHR$(31);
3790 IF MID$(A$,2,1)="K" AND (C > 1) THEN PRINT CHR$(29);
3800 IF MID$(A$,2,1)="M" AND (C < 80) THEN PRINT CHR$(28);
3810 IF (RL = 20) AND (MID$(A$,2,1)="P") THEN GOSUB 4080: REM* PAGE AHEAD
3820 IF (RL = 1) AND (MID$(A$,2,1)="H") THEN GOSUB 4010: LOCATE 20,1: REM* PAGE
BACK
3830 LET RL = R
3840 RETURN
3850 REM
3860 REM ***  GRABAR EN ARRAY ***
3870 REM
3880 R = CSRLIN
3890 C = POS(0)
3900 IF (R = 20) AND (C = 80) AND (P = 3) THEN GOTO 4000
3910 IF A$ = CHR$(13) THEN GOTO 3970
3920 IF A$ = CHR$(9) THEN GOSUB 4440: GOTO 4000
3930 IF C = 80 THEN GOSUB 4150: GOTO 4000
3940 IF A$ = CHR$(8) THEN MID$(T$(P,R),C,1) = " " ELSE MID$(T$(P,R),C,1)= A$
3950 IF A$ <> CHR$(8) THEN PRINT A$;
3960 GOTO 3990
3970 IF R = 20 THEN GOSUB 4080 ELSE R = R + 1:LOCATE R,1,1
3980 R = CSRLIN
3990 LET RL = R
4000 RETURN
4010 REM
4020 REM *** UNA PAGINA HACIA ATRAS ***
4030 REM
4040 IF P = 1 THEN GOTO 4070
4050 P = P - 1
4060 GOSUB 2570
```

```
4070 RETURN
4080 REM
4090 REM *** UNA PAGINA HACIA ADELANTE ***
4100 REM
4110 IF P = 3 THEN GOTO 4140
4120 P = P + 1
4130 GOSUB 2570
4140 RETURN
4150 REM
4160 REM *** EDICION DE LINEA ***
4170 REM
4180 MID$(T$(P,R),80,1) = A$
4190 IF A$ = " " THEN R = R + 1: LET RL = R: LOCATE R,1,1:GOTO 4410
4200 LET I = 80
4210 LET X$ = "N"
4220 WHILE X$ = "N"
4230    I = I - 1
4240    IF MID$(T$(P,R),I,1) = " " THEN LET X$ = "F"
4250 WEND
4260 I = I + 1
4270 FOR J = I TO 80
4280    LET E$(J - I + 1) = MID$(T$(P,R),J,1)
4290    MID$(T$(P,R),J,1) = " "
4300 NEXT J
4310 LOCATE R,1: PRINT T$(P,R)
4320 IF R < 20 THEN R = R + 1 ELSE LET R = 1
4330 LET RL = R
4340 I = 80 - I + 1
4350 IF R = 1 THEN GOSUB 4080
4360 LOCATE R,1,1
4370 FOR J = 1 TO I
4380    PRINT E$(J);
4390    MID$(T$(P,R),J,1) = E$(J)
4400 NEXT J
4410 C = POS(0)
4420 C = C - 1
4430 RETURN
4440 REM
4450 REM *** CHEQUEO DEL TAB ***
4460 REM
4470 IF (R = 20) AND (C > 70) THEN LOCATE 20,80 ELSE PRINT A$;
4480 RETURN
```

TECNICAS DE ANALISIS

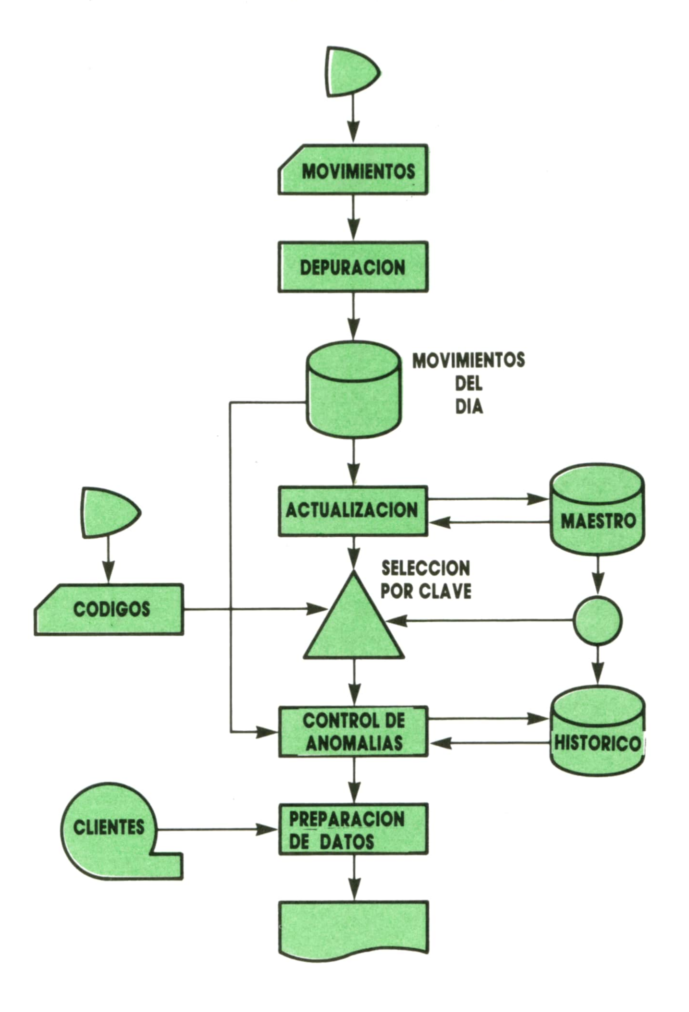
ORGANIGRAMAS Y DIAGRAMAS



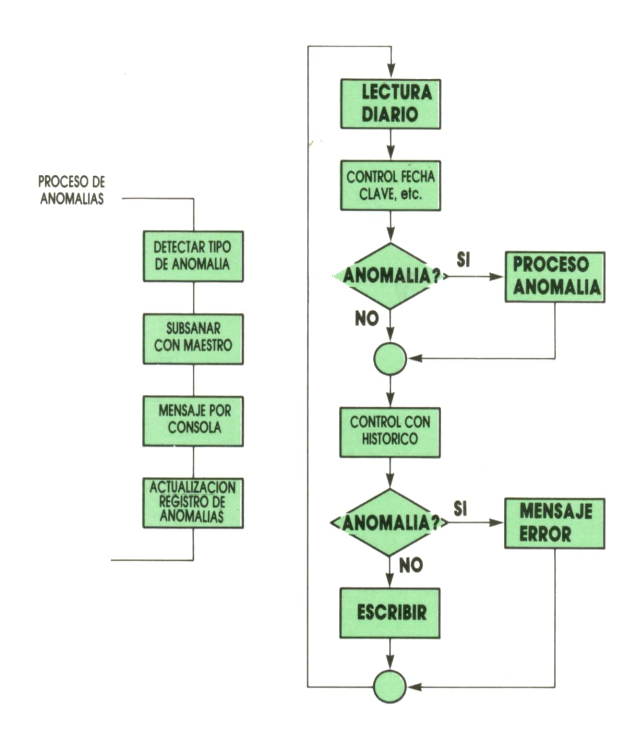
EN toda la tarea de análisis han de buscarse esencialmente dos cualidades: la seguridad en los procesos (controles, examen de todas las condiciones posibles y todos los casos, seguridades de proceso...) y la claridad de exposición de los procesos diseñados. Es importante la claridad tanto por el hecho de que las personas que van a realizar la programación y las que van a utilizarlo posteriormente son distintas (muy comúnmente) de quien diseña el sistema, como por la circunstancia (que se da con cierta frecuencia) de que posteriormente suelen tener que conocer (y quizá modificar o ampliar) el análisis otras personas distintas de quien hizo el primer estudio. Por todo esto es básico que se prepare una documentación exhaustiva del análisis y que, además, sea muy clara. A la claridad de exposición ayuda enormemente la utilización de normas y formularios preimpresos y el uso de organigramas, diagramas, ordinogramas... y, en general, cualquier representación gráfica y visual de los procesos a realizar.

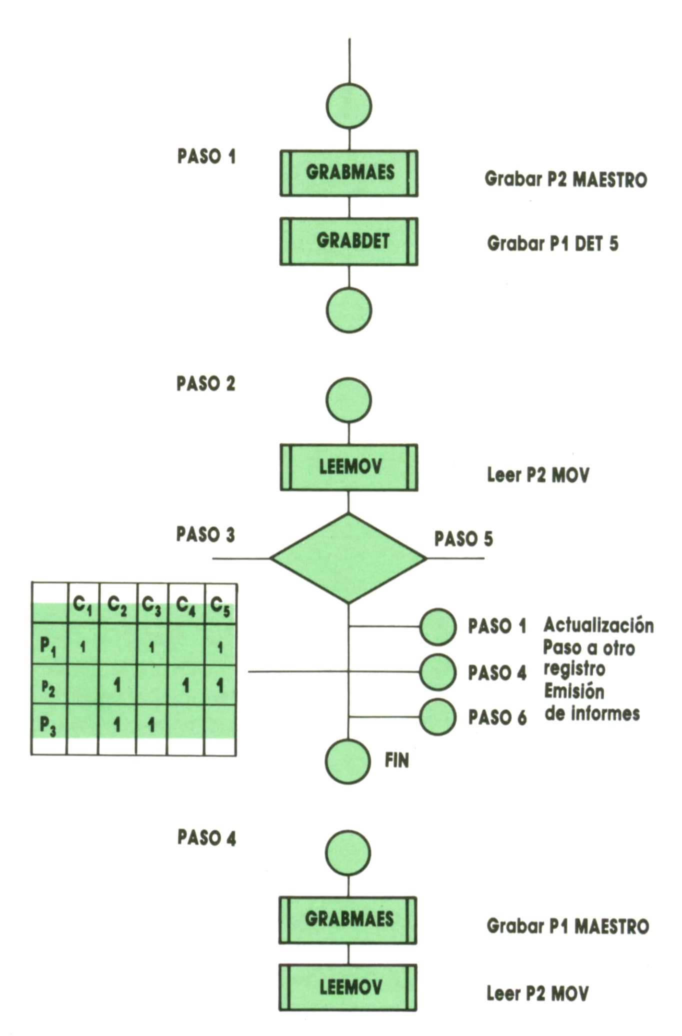
Para la representación gráfica se pueden utilizar diversos tipos de esquemas y no existe unanimidad respecto de qué tipo es el óptimo para cada descripción (ni incluso, en el nombre de cada tipo de representación).

Nosotros en lo que sigue aludiremos a tres tipos de esquema de representación: organigramas, ordinogramas y diagramas de flujo. Se entiende por organigrama una representación bidimensional de la estructura general de un proceso o un sistema con sus relaciones y vinculaciones. (Por supuesto, esta definición es válida para los organigramas utilizados en el análisis informático; la definición, el concepto e, incluso, la simbo-



logía son distintos si se trata de organigramas que reflejan una estructura empresarial, el funcionamiento de un organigrama cualquiera...). Por ordinograma entendemos una representación, del estilo de la anterior, en que se subraya la





conexión de los diferentes procesos para seguir su secuencia, y las relaciones entre una actividad y otras vinculadas a ella, los datos que deben pasar de un proceso a otro, etc. Por último, en el diagrama de flujo (representación lineal del proceso) lo importante es subrayar el «avance» de la información y la evolución y transformación de los datos a lo largo del proceso, sin reflejar la estructura general del proceso ni los elementos que intervienen en el programa.

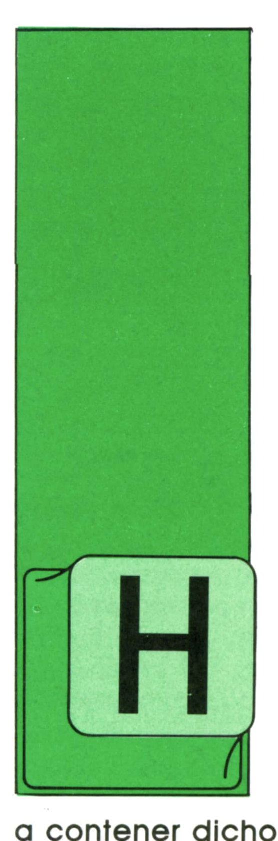
La concreción de la información aumenta a medida que seguimos los pasos indicados: organigrama, ordinograma y diagrama de flujo. Dentro de los organigramas, cabe la preparación de varios tipos, dependiendo de si queremos representar el proceso en su conjunto o una parte de él, toda una aplicación o unos programas; y así, se suelen preparar «organigrama general de la aplicación», «organigrama de programa», «organigrama de detalle de un proceso», etc.

La utilización que se suele dar a los símbolos es la siguientes:

Símbolo de proceso. Para representar un tratamiento cualquiera o una operación que no tenga otro símbolo previsto.

Símbolo de operación manual. Se refiere a los procesos fuera de línea (off-line) que haya que realizar. Es importante reseñarlos, pues, al ser su velocidad normalmente lenta respecto al proceso por el ordenador y/o los periféricos deben ser tenidos muy en cuenta.

Operación auxiliar. Normalmente se representan con este símbolo las operaciones realizadas fuera de línea por otras máquinas auxiliares o mediante otros procesos previos, que deben estar disponibles, pero que no son estrictamente intervenciones del operador y no tienen, por tanto, que consumir tiempo.

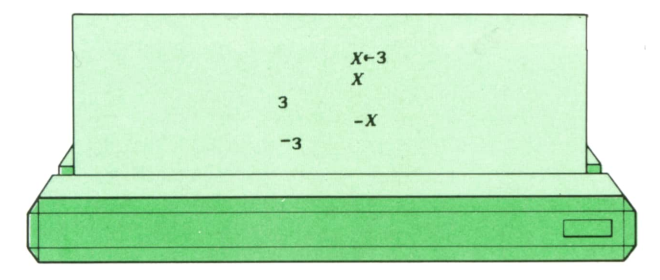


TECNICAS DE PROGRAMACION

EXPRESIONES

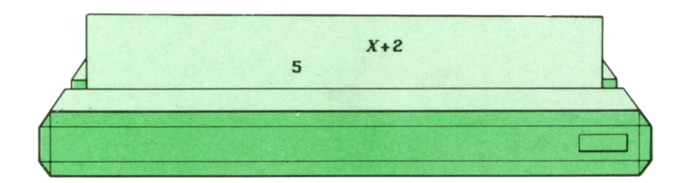
HEMOS visto hasta ahora cómo se definen los datos de un programa (especificando su estructura y su tipo) y cómo se pueden asignar valores a las variables que van a contener dichos datos. Pero para que nuestros programas sirvan para algo, no basta con disponer de uno o varios datos: tenemos también que realizar operaciones con ellos. Estas operaciones se llevan a cabo por medio de expresiones.

Una «expresión» es una fórmula que realiza una o varias «operaciones» con uno o más «datos» produciendo un nuevo dato (el «resultado» de la expresión). En las expresiones existen, por tanto, dos clases de términos diferentes: las «operaciones» y los «datos». La expresión más sencilla es la que realiza una sola operación con un solo dato, obteniendo un solo resultado. Por ejemplo, podemos tomar el valor de una variable y cambiarlo de signo. Veamos cómo se hace esto en APL:

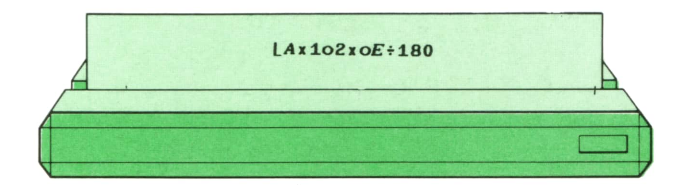


En nuestro ejemplo, el dato es X (cuyo valor es 3) y la operación es el cambio de signo (representado por el signo menos). El resultado de la operación es, naturalmente, el valor −3. Por tanto, la fórmula −X es una expresión en el lenguaje APL.

Otras expresiones, también muy sencillas, realizan una sola operación sobre dos datos diferentes. Por ejemplo, la fórmula X+2 , válida en casi todos los lenguajes de ordenador, obtiene un resultado que supera en dos unidades al valor de X:

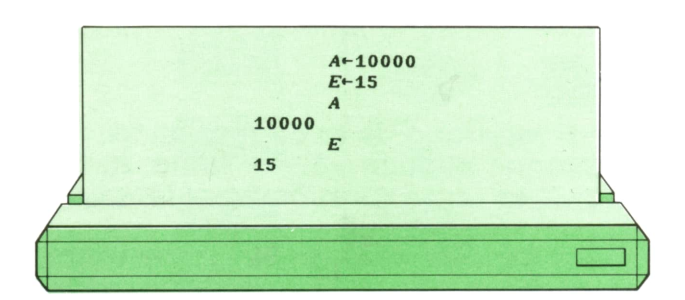


Este último ejemplo nos indica que los datos que forman parte de una expresión pueden ser, indistintamente, constantes (como el 2) o variables (como la X). Además puesto que el resultado de una expresión es también un dato, también puede formar parte de otra expresión. Esto significa que las expresiones pueden concatenarse unas con otras, actuando cada una sobre el resultado de la anterior, hasta llegar a ser tan complicadas como se desee. Por ejemplo:

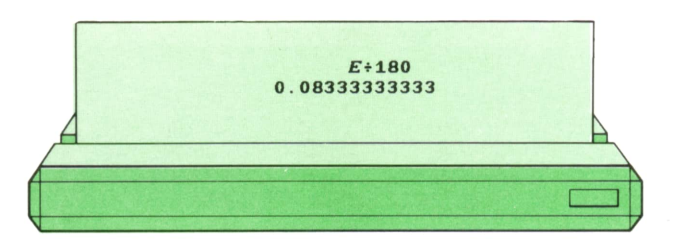


La expresión anterior calcula la distancia a la que cae la bala de un cañón si

el alcance máximo del arma (la distancia alcanzada por la bala cuando se dispara el cañón con un ángulo de 45°) es «A» metros y el ángulo de elevación del cañón es igual a «E» grados. Para comprenderla, vamos a descomponerla en partes elementales, cada una de las cuales es una expresión simple que realiza una sola operación sobre uno o dos objetos. En lo que sigue, supondremos que el valor de A es igual a 10.000 metros y que E es un ángulo igual a 15 grados.

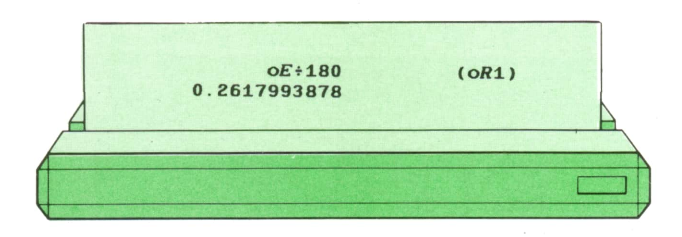


1. Comencemos el análisis de la expresión anterior por la derecha. La primera expresión parcial que encontramos es la siguiente:



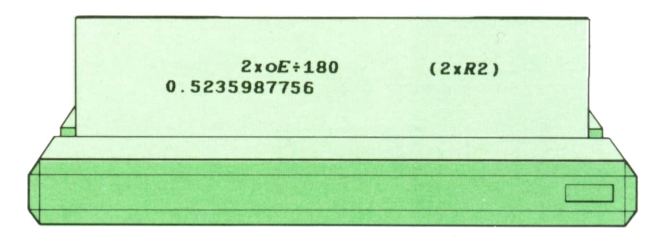
que no hace otra cosa que dividir el valor de la variable E entre 180. Se trata por tanto, de una expresión compuesta por una sola operación que actúa sobre dos datos, y cuyo resultado es, naturalmente, igual a 1/12. A este resultado lo llamaremos R1.

2. A continuación, tendremos que realizar la operación siguiente sobre el resultado anteriormente calculado:



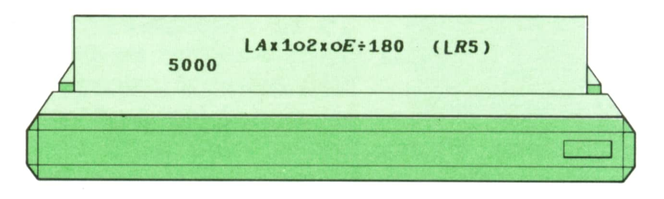
En APL, la operación representada con un círculo y aplicada a un solo dato, obtiene como resultado el producto del dato al que se aplica por el número «pi» (3,141592654). Por tanto, el número obtenido, que llamaremos R2, será igual a pi/12.

3. Viene ahora la tercera operación, que consiste en multiplicar por 2 el resultado R2 obtenido anteriormente:



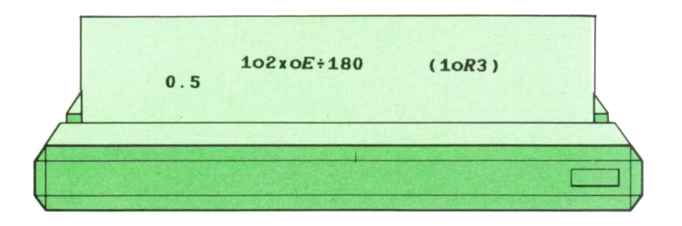
Llamaremos R3 al resultado obtenido.

4. La cuarta operación se representa así:



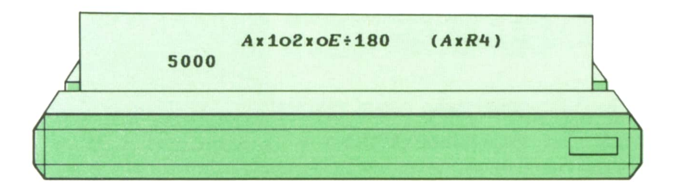
En APL, la operación representada por un círculo y aplicada a dos datos representa las funciones trigonométricas e hiperbólicas. El número de la izquierda indica cuál de estas funciones se va a realizar. Un 1 representa el seno, un 2 el coseno, un 3 la tangente, etc. El dato que se coloca a la derecha es el valor del ángulo, expresado en radianes, sobre el que queremos calcular la función trigonométrica correspondiente. En nuestro caso, como la expresión se reduce a «1 círculo R3», queremos calcular el seno de R3 (pi/12 radianes o 30°), que es exactamente igual a 1/2. Llamaremos R4 al nuevo resultado obtenido.

5. La operación siguiente que debemos realizar es una simple multiplicación.



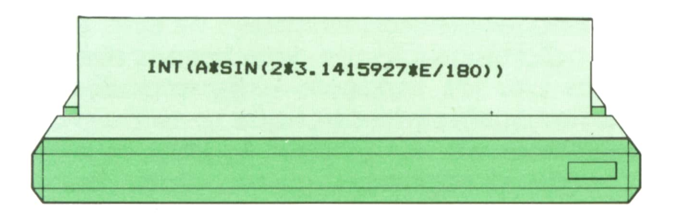
Llamaremos R5 al resultado de multiplicar A (10000) por R4 (0,5) que, naturalmente, es igual a 5000.

6. Llegamos al fin a la última operación de nuestra expresión:



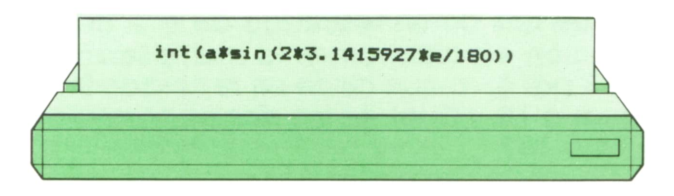
En APL, el símbolo representado por un ángulo recto representa la funcion «parte entera», que calcula la parte entera del dato al que se aplica. En nuestro caso, la parte entera de R5 resulta también ser igual a 5000, pues R5 es entero. Y éste es, precisamente, el resultado de la expresión completa. Esto significa que si disparamos un cañón cuyo alcance máximo es igual a 10000 metros con un ángulo de 15°, la bala alcanzará sólo una distancia de 5000 metros, exactamente igual a la mitad del alcance máximo del arma.

Veamos cómo se escribiría la misma expresión en el lenguaje BASIC:



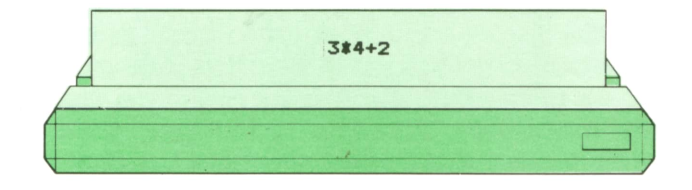
Observaremos que la expresión BASIC se parece mucho a la expresión APL, aunque existen algunas diferencias. En primer lugar, en BASIC se utilizan palabras reservadas en inglés para representar algunas operaciones, que reciben el nombre de «funciones incorporadas». En nuestro ejemplo aparecen dos: SIN (del inglés «sine»), que calcula el seno de un ángulo expresado en radianes, e INT (del inglés «INTEGER»), que calcula la parte entera de un número con decimales. Estas funciones incorporadas se caracterizan porque el dato al que se aplican debe ir entre paréntesis. Las otras diferencias consisten en la utilización de símbolos distintos para las operaciones aritméticas elementales (un asterisco para la multiplicación y una línea inclinada para la división) y en la necesidad de especificar explícitamente el valor de «pi».

Veamos, por último, la versión PASCAL de esta misma expresión:



Como se ve, es idéntica a la expresión BASIC. En realidad, la construcción de expresiones es la parte de los lenguajes de programación en que éstos difieren menos entre sí. Podrán utilizarse unos símbolos u otros, o bien palabras reservadas, pero la expresión en sí suele parecerse bastante de un lenguaje a otro. Unicamente en lenguajes como APL, mucho más potentes que los lenguajes de programación corrientes de la tercera generación, encontramos operaciones (como la conversión de base de sistema de numeración, el producto de matrices o la inversión matricial) que se expresan con un solo símbolo, mientras en otros lenguajes (como BASIC o PASCAL) precisan para su cálculo de programas completos formados por numerosas instrucciones ejecutables.

Acabamos de ver que una expresión compleja puede descomponerse siempre en dos o más expresiones más sencillas. Sin embargo, aun no hemos dicho nada sobre el orden en que se realizarán las operaciones, o cómo puede conseguirse que unas tengan lugar antes de otras. Como ejemplo, consideremos la expresión BASIC siguiente,



que se compone de dos operaciones (una multiplicación y una suma) que actúan sobre un total de tres datos (los números 3, 4 y 2). ¿Cuál es la operación que debemos realizar primero? ¿La multiplicación o la suma? El resultado final sería totalmente diferente en uno y otro caso. En efecto, si empezamos por la multiplicación, la primera operación a realizar será 3 por 4, cuyo resultado es 12. En la segunda operación, habrá que sumar 12 más 2, dando un resultado final de 14. Sin embargo, si empezáramos por la suma, comenzaríamos por calcular 4 más

2, que nos da un resultado de 6. A continuación efectuaríamos la multiplicación de 3 por 6, lo que daría un resultado final igual a 18. ¿Cuál de los dos es correcto? ¿14 ó 18?

No existe respuesta a la pregunta anterior. En realidad, los dos son correctos, dependiendo de las reglas del juego. En algunos lenguajes de programación, como en BASIC y PASCAL, se admite que unas operaciones tienen precedencia intrínseca sobre otras. En particular, en circunstancias como las anteriores, la multiplicación tiene siempre precedencia respecto a la suma. Por tanto, se realizará antes, y el resultado de la expresión será igual a 14. En estos lenguajes, existe toda una jerarquía de operaciones que hay que aprender de memoria y tener presente a la hora de escribir programas que contengan expresiones con operaciones múltiples.

Veamos la jerarquía de las operaciones en el lenguaje BASIC:

1. Potenciación, representada por el símbolo ^.
2. Cambio de signo, representado por el signo – aplicado a un solo dato.
3.. Multiplicación (*) y división (/), indistintamente. Cuando coinciden varias multiplicaciones o divisiones en la misma expresión, la que se encuentra más a la izquierda se efectúa primero.
4. División entera, representada por el símbolo \.
5. Resto de la división entera, representada por la palabra reservada «MOD». Esta operación, junto con las seis operaciones lógicas indicadas más abajo, es una excepción a la regla de que las funciones representadas por palabras reservadas tienen siempre sus datos entre paréntesis. El resto de dividir 8 entre 3 se representa como «8 MOD 3».
6. Suma (+) y resta (–) aplicadas a dos datos, indistintamente. Cuando coinciden varias sumas y restas en la misma expresión, la que se encuentra más a la izquierda se efectúa primero.
7. Comparaciones de valor, representadas por los símbolos = (igualdad), < (menor que), > (mayor que), <= o =< (menor o igual que), >= o => (mayor o igual que) y <> o >< (desigualdad).
8. Operación lógica NOT.
9. Operación lógica AND.
10. Operación lógica OR.
11. Operación lógica XOR.
12. Operación lógica EQV.
13. Operación lógica IMP.

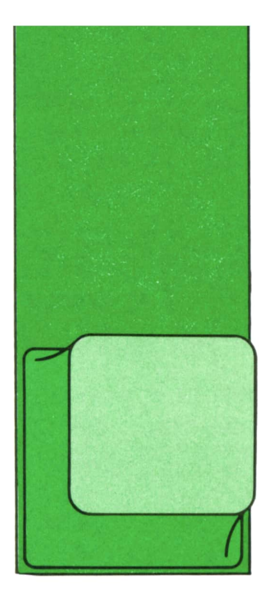
Explicaremos con detalle las operaciones de comparación y las operaciones lógicas un poco más adelante.

Veamos ahora la jerarquía de las operaciones en el lenguaje PASCAL:

1. Cambio de signo, representado por el signo – aplicado a un solo dato.
2. Operación lógica NOT.
3. Multiplicación (*), división (/), división entera (DIV), resto de la división entera (MOD) y operación lógica AND, indistintamente. Cuando coinciden varias de estas operaciones en la misma expresión, la que se encuentra más a la izquierda se efectúa primero.
4. Suma (+) y resta (–) aplicadas a dos datos, así como las operaciones lógicas OR y XOR, indistintamente. Cuando coinciden varias de ellas en la misma expresión, la que se encuentra más a la izquierda se efectúa primero.
5. Comparaciones de valor, representadas por los símbolos = (igualdad), < (menor que), > (mayor que), <= (menor o igual que), >= (mayor o igual que) y <> (desigualdad).

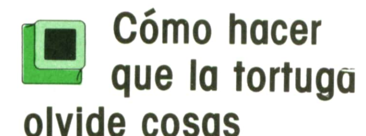
Como puede observarse, el orden de precedencia de PASCAL se parece al de BASIC, pero no es idéntico, pues existen pares de operaciones cuya precedencia es inversa.

En APL existe un número tan grande de operaciones posibles, que sería muy difícil aprender de memoria una tabla de precedencias. Por tanto, se decidió que todas las operaciones tendrían la misma precedencia, y que se ejecutaría primero aquella que estuviera situada más a la derecha. En el ejemplo dado al principio de este capítulo hemos aplicado precisamente, y en forma sucesiva, esta regla de precedencia. También es de notar que en APL la expresión 3×4+2 (correspondiente a la expresión BASIC o PASCAL 3*4+2 dada más arriba) tiene un valor igual a 18, pues la suma se realizará antes que la multiplicación, por estar a su derecha.



LOGO

ALGO MAS SOBRE PROCEDIMIENTOS



Cómo hacer que la tortuga olvide cosas

Si al definir un procedimiento, es decir, enseñar a la tortuga a hacer una cosa nueva para ella y mandarle que lo ejecute nos damos cuenta de que no dibuja lo que nosotros queremos, significa que nos hemos equivocado al escribir algo y tenemos que cambiarlo.

Por ahora, lo que vamos a hacer es decirle que se olvide de eso nuevo que le hemos enseñado y definir el procedimiento otra vez.

Para borrar un procedimiento, es decir, que la tortuga se olvide de cómo se hace, usaremos el comando:

BORRA "nombre

o en abreviatura

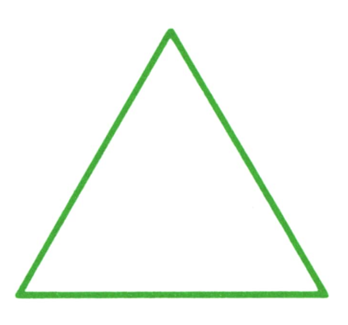
BO "nombre

siendo **nombre** el nombre del procedimiento que queremos borrar.

Si en lugar de querer borrar un solo procedimiento, deseamos borrar varios pondremos sus nombres en una lista:

BO (nombre1 nombre2 ... nombren)

Veamos un ejemplo. Supongamos que queremos dibujar un triángulo así:



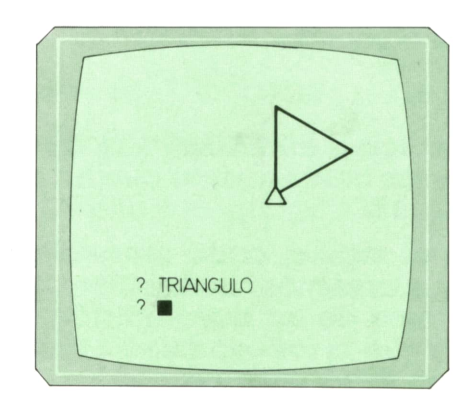
Si definimos un procedimiento:



y mandamos a la tortuga que lo ejecute:

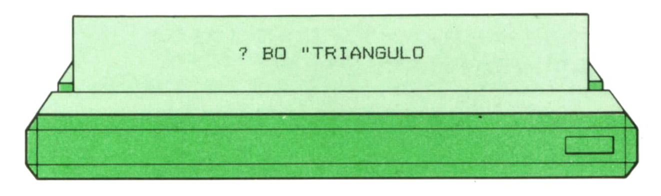
? TRIANGULO

nos saldrá:



que no es lo que nosotros queremos.

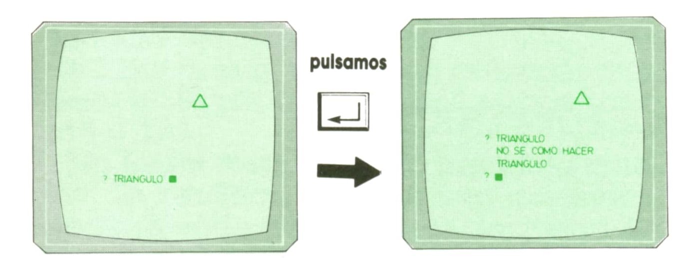
Por tanto, nuestro procedimiento no nos vale y tenemos que borrarlo de la memoria de la tortuga:



Para comprobar que, efectivamente, lo ha olvidado si le decimos ahora:

? TRIANGULO

nos responderá con un mensaje de error:



Entonces, en este momento podemos enseñarle de nuevo a hacer el triángulo, pero correctamente:



Por último, si en lugar de querer borrar uno o varios procedimientos, deseamos eliminar todos, no hace falta dar cada uno de sus nombres, sino que podemos utilizar el comando

BOTODO

que le indica a la tortuga que se olvide de todas las cosas nuevas que ha aprendido.

Como es lógico, cada procedimiento nuevo que creemos ha de tener un nombre diferente de los que ya están definidos para que la tortuga pueda distinguirlos y no se arme un lío.

Por eso, en un momento determinado (sobre todo cuando ya hemos definido muchos procedimientos) nos puede interesar conocer los nombres de todas las cosas nuevas que le hemos enseñado a la tortuga.

El comando que hace que la tortuga nos muestre todos los nombres de los procedimientos definidos es

IMTS

abreviatura de IMprime TítuloS.

Así, por ejemplo, si tenemos definidos los siguientes procedimientos: triángulo, cuadrado, casa y figura, al escribir:

?IMTS

la tortuga responderá:

PARA TRIANGULO
PARA CUADRADO
PARA CASA
PARA FIGURA
?

Por otro lado, es posible que, a veces, nos interese ver no sólo los nombres de los procedimientos que hemos definido, sino también su contenido, es decir, su definición en sí.

Para ello, disponemos de dos comandos. Si deseamos ver la definición de un procedimiento usaremos el comando

IM "nombre

siendo IM la abreviatura de IMprime y nombre el nombre del procedimiento.

Por el contrario, si queremos ver la definición de varios procedimientos a la vez, utilizaremos el comando

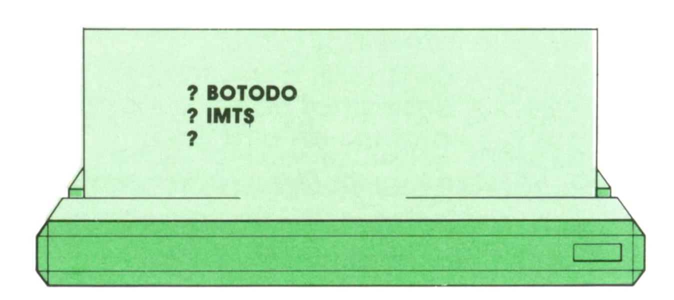
IM (nombre1 nombre2 ... nombren)

es decir, pondremos en una lista sus nombres.

Por último, si lo que queremos hacer es ver la definición de todos los procedimientos que hayamos creado, el comando será

IMTODO

Como es lógico, en el caso de que no le hayamos enseñado nada nuevo a la tortuga o le hayamos mandado que lo olvide todo mediante el comando BOTODO, al pedirle que nos muestre los nombres de procedimientos o su definición no nos contestará nada. Por ejemplo:



Diferentes pantallas

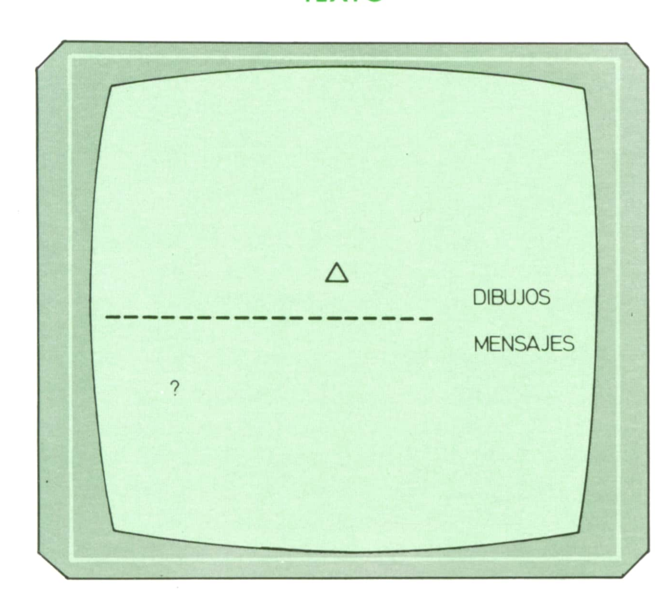
En el caso de que tengamos muchos procedimientos, cuando la tortuga escriba sus nombres, al darle el comando IMTS, no cabrán todos en el espacio de pantalla del que ella dispone para dar-

nos mensajes, que es mucho más pequeño que el espacio reservado para hacer dibujos.

Lo mismo nos puede ocurrir cuando queramos ver la definición de uno o varios procedimientos.

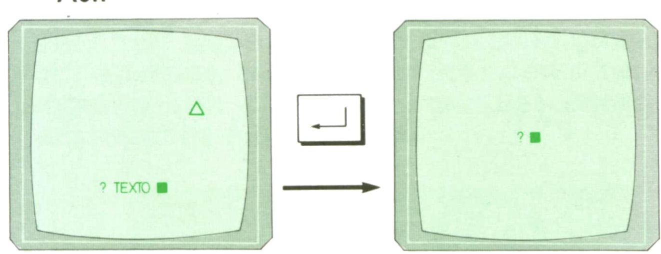
Para que nos quepa en la pantalla toda la información que le podemos pedir a la tortuga, nos convendría que la parte de mensajes fuera más grande. Para ello, existe el comando

TEXTO



que hace que en la pantalla no haya una parte para dibujos.

Así:





Ahora, al decirle IMTS, todos los nombres de procedimientos cabrán en la pantalla y podremos leerlos mejor.

Cuando queramos volver a tener la pantalla dividida en dos partes, utilizaremos el comando

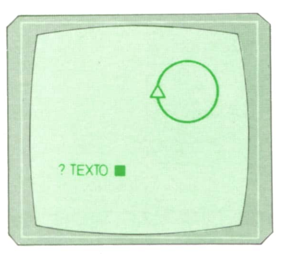
PANTALLAMIXTA

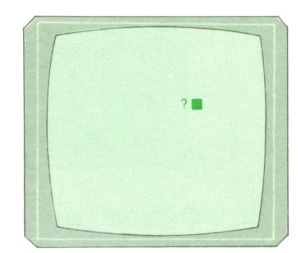
o su abreviatura

PM

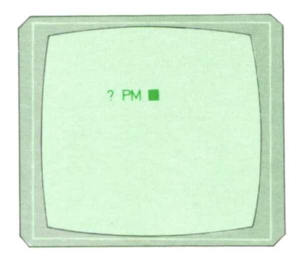
Si usamos este comando, el aspecto de la pantalla será el mismo de siempre pero se habrá borrado todo lo que hubiésemos escrito o dibujado anteriormente.

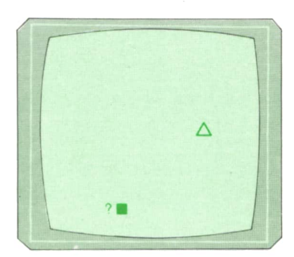
Vamos a probarlo. Supongamos que hemos dibujado una circunferencia:





Si escribimos:

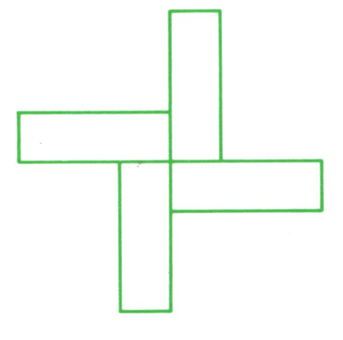




Cuando la pantalla aparece dividida en dos partes (dibujos y texto) se dice que está en MODO GRAFICO y cuando sólo se puede escribir en ella, pero la tortuga no pinta, se dice que está en MODO TEXTO. Resulta mejor utilizar este segundo modo cuando queramos que la tortuga ejecute órdenes que no lleven consigo el que realice dibujos.

Os proponemos

1. Enseña a la tortuga a dibujar un molinillo:



2. Puedes enseñar a la tortuga a dibujar diferentes estrellas. Recuerda que la fórmula para las estrellas es:

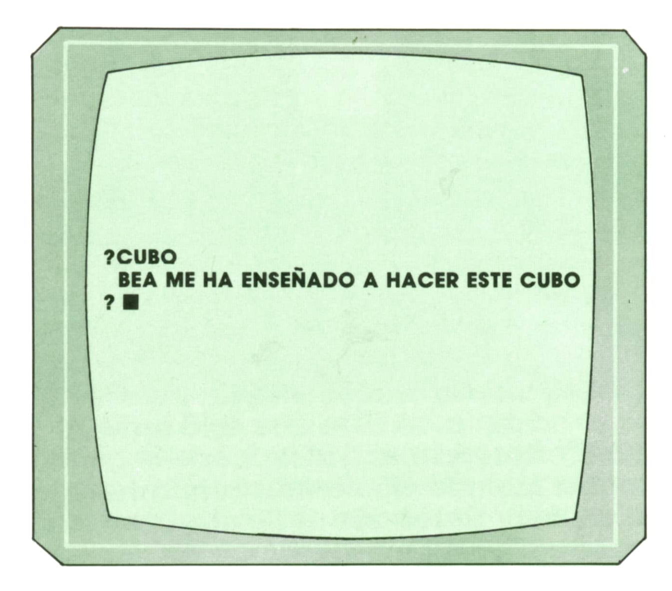
REPITE num. puntas (AV longitud RT total/num.puntas)

donde longitud lo decides tú según quieras estrellas grandes o pequeñas.

Te damos algunos valores para el número de puntas y el total:

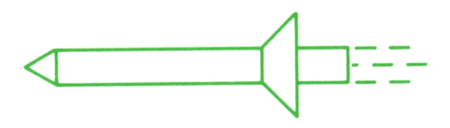
Núm. puntas	Total
9	720
9	1440
30	4680

3. Define un procedimiento que al ejecutarlo dé como resultado lo siguiente:

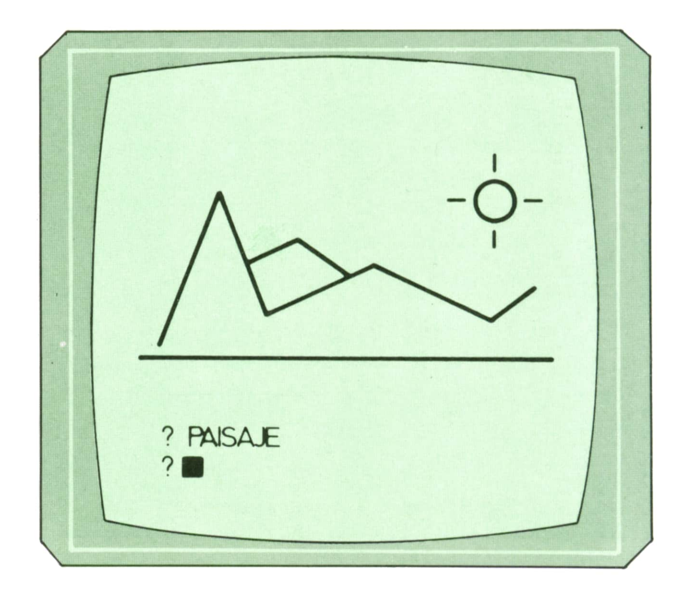


En lugar de BEA, puedes poner tu nombre.

4. A ver si eres capaz de que la tortuga aprenda a dibujar un cohete.



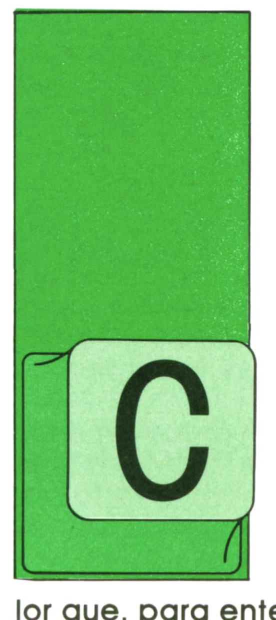
5. Incluso puedes lograr que la tortuga dibuje un paisaje como éste:



PASCAL



Funciones



CUANDO empezamos a hablar de procedimientos y funciones, dijimos que estas últimas son como los primeros, pero con la particularidad de que devuelven un valor que, para entendernos, se podría decir que pasa a ocupar el lugar del nombre de la función en el punto en que fue llamada.

Este valor puede ser de cualquier tipo escalar, es decir, valores simples como INTEGER, CHAR, BOOLEAN o de tipos definidos por enumeración o subrangos. También está admitido un tipo que todavía no hemos estudiado: el tipo REAL. En general, se dice que una función es de tal tipo cuando el dato que devuelve es de ese tipo.

Ya conocemos un ejemplo de función predefinida de tipo Boolean: la función ODD. Para utilizarla no se necesita conocer nada sobre cómo ha sido programada. Además, se puede utilizar como parámetro con ella tanto variables como expresiones o constantes.

En esta misma línea, es una buena práctica escribir las funciones de manera que se puedan utilizar de modo similar a ODD, es decir, absteniéndose de utilizar en ellas variables globales y no utilizando el paso de parámetros por nombre, de manera que la única consecuencia de su utilización esté en el resultado devuelto.

La única diferencia entre los procedimientos normales y las funciones a la hora de escribirlos está en la cabecera. En las funciones se utiliza la palabra reservada FUNCTION en lugar de PROCEDURE; además, a continuación del nombre de la función (o a continuación de la definición de la lista de parámetros, caso de existir éstos) se debe indicar el tipo de resultado que devuelve.

Por otra parte, cuando ya se haya obtenido el resultado, éste debe asignarse al nombre de la función como si de una variable se tratase.

Como ejemplo, vamos a escribir un programa que presente en pantalla el cubo de los diez primeros números naturales. Para ello utilizaremos la función CUBO cuyo parámetro es INTEGER y que devuelve un resultado del mismo tipo:

```
program SacaCubos;
 var
  Ene: integer;

 function Cubo (X: integer): integer;
  (* Devuelve el cubo de X *)
  begin
   Cubo:= X * X * X
  end;

begin    (* Aquí comienza el programa principal *)
 for Ene:=1 to 10 do
  writeln ('El cubo de ',Ene:3,' es ',Cubo (Ene):6)
end.
```

La función CUBO así escrita es directamente utilizable por otros programas sin más que copiar la parte del programa PASCAL que le corresponde. El resultado devuelto es como si estuviera en el sitio desde donde ha sido llamada, por lo que podría aparecer en cualquier lugar en que pudiera figurar un valor entero; por tanto, formas correctas de utilizarla serían:

```
A:= Cubo (2) - 3;                 ( • Guarda 5 en A •)
B:= Cubo (A-1);                   ( • Guarda 4 al cubo en B •)
writeln (Cubo (Cubo (3) - 17));   ( • Presenta 1000 en pantalla •)
```

Veamos otro ejemplo. Supongamos que la función ODD no existiera y que hubiera que escribirla; el parámetro es de tipo INTEGER y se devuelve el valor de tipo Boolean TRUE o FALSE según que aquél sea o no impar.

Para saber si un número es impar o no, una posible manera es comprobar si el resto de dividirlo por dos es uno o cero:

```
function Odd (N: integer): boolean;
 begin
  Odd:= (N mod 2 = 1)
 end;
```

Esta función se utilizaría exactamente igual que la original. Aunque un procedimiento o función esté predefinido, como es el caso de ODD, podemos hacer nuestra propia definición, que será la que realmente se emplee (ODD no es una palabra reservada, sino un identificador predefinido).

Un último ejemplo. En el procedimiento LeeColor que escribimos anteriormente podría suceder que la letra tecleada fuese minúscula, con lo que en las instrucciones IF habría que poner:

```
if (letra='R') or (Letra='r') then C:= Rojo ...
```

y análogamente en la instrucción de asignación a la variable OK para permitir tanto mayúsculas como minúsculas.

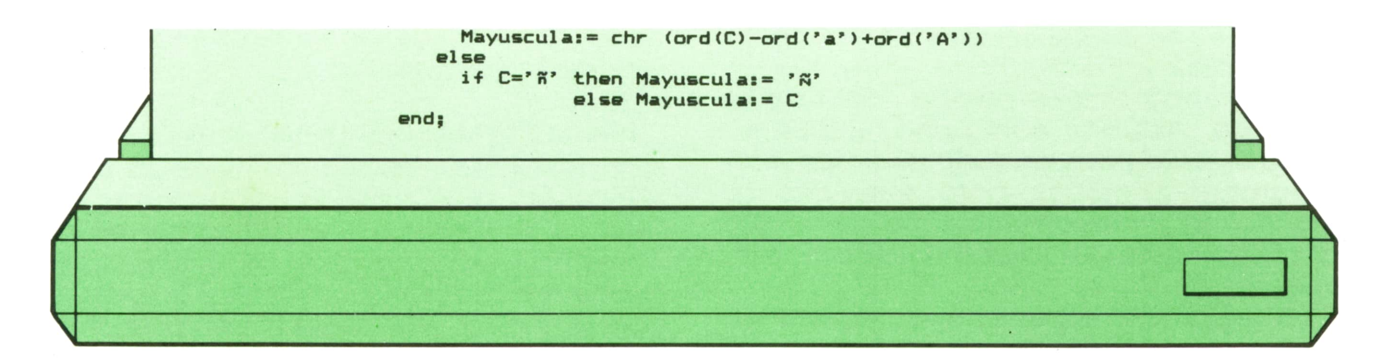
En lugar de ello, tras leer la tecla pulsada la pasaremos a mayúscula si es que fue una minúscula. Para hacerlo, utilizaremos la función MAYUSCULA cuyo parámetro es un carácter y que devuelve el mismo carácter excepto en el caso en que aquél sea una letra minúscula, en que devuelve la mayúscula equivalente:

```
function  Mayuscula (C: char): char;
 begin
    (* Si C es minúscula, está entre 'a' y 'z' *)

   if ('a' <= C) and (C <= 'z') then
       Mayuscula:= chr  (ord(C)-ord('a')+ord('A'))
     else
       Mayuscula:= C
 end;
```

La operación **ord(C)-ord('a')** proporciona 0, 1, 2,... según que C sea 'a', 'b', 'c'... Si a este número le sumamos ord ('A'), obtendremos el número de orden de 'A', 'B', 'C'..., con lo que utilizando la función CHR se consigue la letra mayúscula equivalente. Esto, claro está, siempre que la letra sea una de las 26 del alfabeto inglés. Si además se necesitara la letra Ñ:

```
function  Mayuscula (C: char): char;
 begin
  if ('a' <= C) and (C <= 'z') then
```



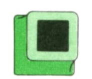
Esta función podría incorporarse a su vez como una función del procedimiento LeeColor en el sitio que le corresponde, es decir, tras su zona de definición de datos y antes de sus instrucciones:

```
procedure LeeColor (var C: ColorPrimario_t);
(* Lee de teclado un color *)
   var Letra: char; Ok: boolean;
(*----------------------------------------*)
  function  Mayuscula (C: char): char;
   begin
    if ('a' <= C) and (C <= 'z') then
        Mayuscula:= chr (ord(C)-ord('a')+ord('A'))
      else
        if C='ñ' then Mayuscula:= 'Ñ'
                 else Mayuscula:= C
   end;
(*----------------------------------------*)
 begin
  (* Pedir letra del color: *)
  write ('Color? (R,V,A) ');
  repeat
    readln (Letra);
             (* pasamos la letra a mayúscula: *)
    Letra:= Mayuscula (Letra);
    Ok:= (Letra='R') or (Letra='V') or (Letra='A');
    if not Ok then writeln ('Letra no válida.')
  until Ok;

  if Letra = 'R' then C:= Rojo
    else if Letra = 'V' then C:= Verde
     else C:= Azul
 end;
```

Por casualidad, tanto la variable de LeeColor como el parámetro de Mayuscula tienen el mismo nombre, C; podrían llamarse de distinta manera, pero da lo mismo. La variable de LeeColor es global para Mayuscula, pero como se empieza a buscar primero entre las locales propias, la que se utiliza en la función es la correcta. No obstante, si con ello se evitan confusiones, no hay que dudar en cambiar los nombres.

Por otra parte, en PASCAL, como en la mayoría de lenguajes de programación, no suele estar permitido el empleo de letras acentuadas o eñes al escribir identificadores, de ahí la ausencia de acento en Mayuscula.



Relaciones entre procedimientos

La función Mayuscula inserta en LeeColor es local de este procedimiento, por lo que sólo puede utilizarse desde las instrucciones de éste. Para el programa principal en que estuviera a su vez inserto el procedimiento, la función es un detalle de las interioridades de LeeColor y

como tal no tiene conocimiento de su existencia.

Podríamos haber escrito Mayuscula fuera de LeeColor como una función independiente para poderla usar desde el programa principal, pero entonces se plantea la siguiente cuestión: ¿va a poder seguir siendo utilizada por el procedimiento?

La regla para saber cuándo puede ser utilizado un procedimiento o función es la siguiente:

«Puede llamarse a un procedimiento o función desde dentro de su inmediato poseedor tomado en su conjunto y siempre desde puntos que se encuentren por detrás de su cabecera.»

Por ello, si Mayuscula fuese una función independiente, podría ser utilizada desde dentro de su inmediato poseedor, es decir, el programa principal. Lo de «en su conjunto» significa que sería utilizable desde el programa principal y desde cualquier procedimiento o función incluido en éste, siempre y cuando se encuentre por detrás de Mayuscula. Así, para que Mayuscula sea utilizable desde LeeColor y cualquier otro punto del programa, bastará con ponerla por separado y delante de aquél:

```
    function  Mayuscula (C: char): char;
     begin
      if ('a' <= C) and (C <= 'z') then
          Mayuscula:= chr (ord(C)-ord('a')+ord('A'))
        else
          if C='ñ' then Mayuscula:= 'Ñ'
                   else Mayuscula:= C
     end;
  (*---------------------------------------*)
  procedure LeeColor (var C: ColorPrimario_t);
  (* Lee de teclado un color *)
     var Letra: char; Ok: boolean;
   begin
    (* Pedir letra del color: *)
    write ('Color? (R,V,A) ');
    repeat
      readln (Letra);
                (* pasamos la letra a mayúscula: *)
      Letra:= Mayuscula (Letra);
      Ok:= (Letra='R') or (Letra='V') or (Letra='A');
      if not Ok then writeln ('Letra no válida.')
    until Ok;

    if Letra = 'R' then C:= Rojo
      else if Letra = 'V' then C:= Verde
       else C:= Azul
   end;
```

Evidentemente, de esta manera LeeColor pierde su «autosuficiencia» (en el sentido de que para utilizarlo no basta con copiarlo, sino que además es necesario que se encuentre presente Mayuscula en nuestro programa), con la compensación de tener a Mayuscula disponible para su uso por otras partes del programa.

Otra pregunta se plantea: si, además de esta función Mayuscula independiente, LeeColor siguiera teniendo su propia función local, ¿cuál de ellas utilizaría el procedimiento?

Al igual que sucede con las variables, cuando el compilador encuentra un nombre de procedimiento o función, busca primero entre los locales del punto en que se encuentra, por lo que LeeColor utilizaría su propia función y el resto del programa utilizaría la independiente.

Veamos otro ejemplo. Supongamos un programa con la siguiente estructura:

```
program Ejemplo;
  (*-----------------------------------------*)
  procedure Gordo;
      var LocalDeGordo: char;
      (*--------------------*)
      procedure Primero;
       begin
         (* Instrucciones de Primero *)
       end;
      (*--------------------*)
      procedure Segundo;
       begin
         (* Instrucciones de Segundo *)
       end;
      (*--------------------*)
   begin
     (* Instrucciones de Gordo *)
   end;
  (*-----------------------------------------*)
  procedure Flaco;
      (*--------------------*)
      procedure Activar;
       begin
         (* Instrucciones de Activar *)
       end;
      (*--------------------*)
   begin
     (* Instrucciones de Flaco *)
   end;
  (*-----------------------------------------*)
begin
(* Instrucciones del programa *)
end.
```

En este programa nos encontramos con lo siguiente:

— El procediminto Gordo, como su inmediato poseedor es el programa principal, puede utilizarse desde cualquier parte, al estar su cabecera por delante de todas las zonas posibles de instrucciones.

— El procedimiento Flaco se encuentra a un mismo nivel que Gordo, por lo que puede utilizarse desde cualquier punto del programa siempre que se encuentre por detrás de su cabecera, es decir, desde cualquier parte menos desde el interior de Gordo.

— El inmediato poseedor de Primero es Gordo, por lo que sólo puede utilizarse desde éste en su conjunto, es decir, desde sus instrucciones y desde las instrucciones de Segundo. Sucede lo mismo con Segundo, con la diferencia de que este último no puede utilizarse desde las instrucciones de Primero, pues no están por detrás de su cabecera.

— Análogamente, Activar sólo puede utilizarse desde el interior de Flaco.

Deliberadamente se ha omitido la posibilidad, no reñida con la regla, de que, por ejemplo, Primero se utilice también desde sus propias instrucciones, o de que entre las instrucciones de Flaco se encuentre una llamada a Activar, entre cuyas instrucciones esté a su vez una llamada a Flaco, etc. Situaciones así, en que un procedimiento se llama a sí mismo de manera directa o indirecta, se dice que son «recursivas», y las estudiaremos en otra ocasión.

La cabecera debe encontrarse por delante del punto de utilización porque una de las características del lenguaje PASCAL es que, sea cual sea el punto en el que aparece un identificador, éste debe haber sido definido previamente. Por ejemplo, el programa puede utilizar una variable que ha sido definida previamente en la zona VAR utilizando un tipo que ha sido definido previamente en las zonas TYPE como sub-rango delimitado por dos constantes definidas previamente en la zona CONST. A la hora de escribir un programa esto no es una gran limitación,

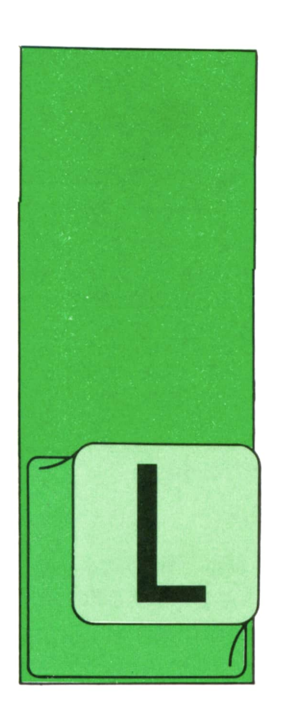
pues no tiene demasiado sentido utilizar algo que todavía no se ha definido.

En definitiva, las ventajas principales de ubicar unos procedimientos dentro de otros son dos:

Por un lado, es posible hacer a estos últimos «autosuficientes», pues contienen todas las funciones, procedimientos y variables necesarios para su funcionamiento, de manera que basta con copiarlos para utilizarlos en otros programas (aunque, en el caso concreto de LeeColor, el tipo ColorPrimario_t debe estar definido en el programa principal).

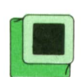
Por otro, permite sustituir los procedimientos generales, disponibles para todo el programa (entre los que se incluyen los predefinidos), por los suyos propios.

Con los procedimientos y funciones de uso general, sin embargo, lo más cómodo será escribirlos de manera independiente para así poderlos utilizar desde cualquier punto, como es el caso de Mayuscula.



OTROS LENGUAJES

LENGUAJE C



Sentencia IF-ELSE

A sentencia *if* nos permite ejecutar una sentencia, si se cumple una condición, como la contraria, a través de *else*.

```
if (expresión)
   sentencia A;
else
   sentencia B;
```

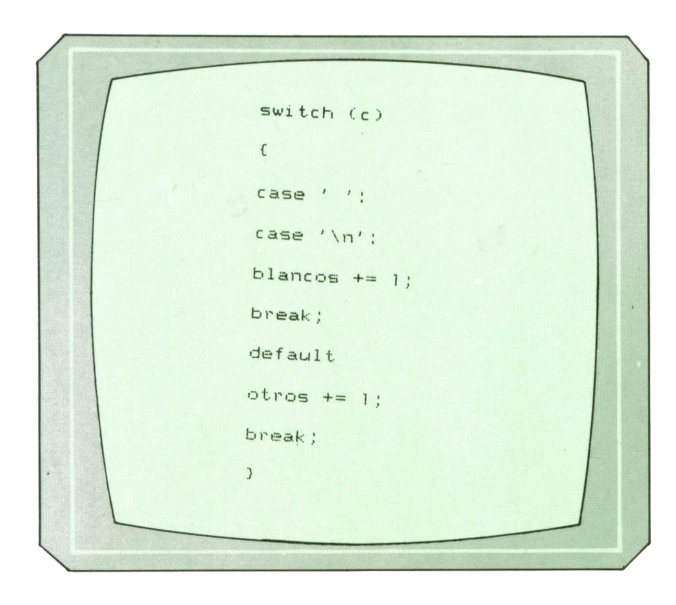


Sentencia SWITCH

La sentencia *switch* va acompañada de una expresión entre paréntesis. La sentencia evalúa la expresión y compara su valor con todos los casos *(case)*. Si alguno de los casos es idéntico al valor de la expresión que acompaña al *switch*, se ejecutará la sentencia etiquetada por *case*.

La línea con *default* se utiliza en el supuesto de que no se satisfaga ninguno de los *case* al compararse con la expresión que acompaña al *switch*.

Veamos un ejemplo.





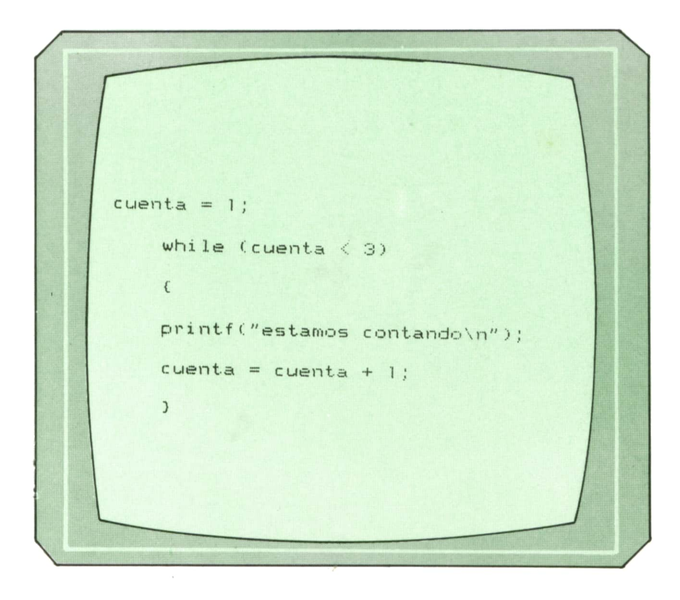
Sentencia WHILE

Es una sentencia que se utiliza para la construcción de bucles. La repetición puede llevarse a cabo sobre una sentencia o bloque de sentencias.

```
while (expresión)
sentencia;
```

Se procede de la siguiente forma: se evalúa la expresión y si resulta ser cierta, se ejecuta la sentencia una y otra vez hasta que la expresión deje de ser cierta.

Veamos un ejemplo de esta sentencia.



Sentencia FOR

La sentencia *for* se utiliza para que una sentencia o bloque de sentencias se ejecute repetidamente en función de tres expresiones separadas por punto y coma.

La sintaxis general de un bucle *for* es:

```
for (expresión1; expresión2; expresión3)
    sentencia;
```

La primera expresión es la de inicialización, la segunda de condición de prueba y la tercera se evalúa al final de cada bucle.

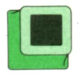


Sentencia DO-WHILE

El bucle *do-while* comprueba la condición después de cada pasada a través de la sentencia o bloque de sentencias. Cuando la condición deja de cumplirse, el bucle finaliza.

La sintaxis utilizada es:

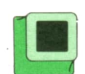
```
do
  sentencia
while (expresión);
```



Sentencias BREAK y CONTINUE

La sentencia "break" fuerza la salida de un bucle *for, while* o *do* de la misma forma que sucedía en *switch.*

La sentencia *continue* hace que el resto de la iteración se ignore, obligando a comenzar una nueva iteración.

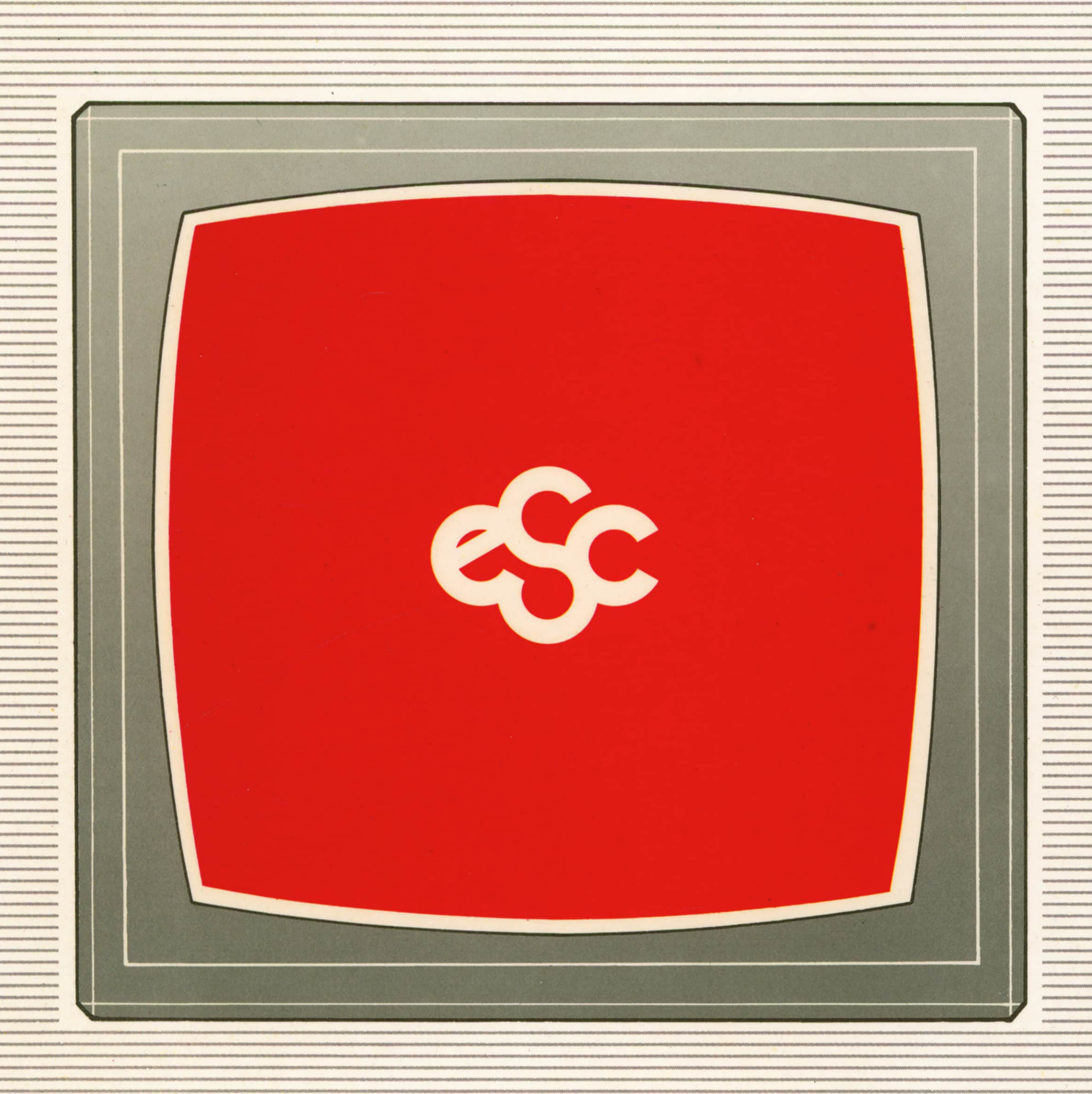


Sentencia GOTO

La sintaxis de la sentencia es:

```
goto etiqueta
```

La etiqueta se rige por las mismas normas que las variables. El uso más normal de la sentencia *goto* consiste en salir de un proceso en una estuctura que esté fuertemente anidada, ya que en este caso la utilización de la sentencia *break* sólo nos llevaría a abandonar la estructura más interna.



▼ EDICIONES ▼ SIGLO ▼ CULTURAL ▼